QCOMPARE(HTML5, QML)

A comparison of UI development technologies.

Casper van Donderen

Abstract

The mobile landscape is constantly evolving, from big carphones to the small and powerful smartphones of today. This evolving landscape also brings new requirements to the mobile user interface.

Nokia has embraced two technologies for mobile application development: QML and HTML. Both technologies can be used to develop modern mobile applications, but the question is: *What is the best technology?*. To find the answer to this question I have researched requirements for mobile applications by reading scientific articles. The next step was researching the features and system requirements of both technologies by studying the technology specifications and developing applications using both technologies.

The main conclusion of the report is: The technologies are similiarly featured with HTML having advantages in the web environment and QML making it possible to develop highly-animated user interfaces.

Both performance-wise and memory-wise QML is clearly the best technology to develop mobile user interfaces. Depending on the requirements regarding external data handling and preference of the developer the choice for HTML can also be made.

Contents

List of Figures **vii**

Introduction **ix**

Problem definition **xi**

Assignment **xiii**
- Sub questions . . . xiii
- Objectives . . . xiii
- Products . . . xiii

Plan of approach **xv**

1 Mobile **1**
- 1.1 Maemo . . . 1
  - 1.1.1 History of Maemo . . . 1
  - 1.1.2 Maemo technologies . . . 2
    - 1.1.2.1 Clutter . . . 2
    - 1.1.2.2 GStreamer . . . 3
    - 1.1.2.3 GTK+ . . . 3
    - 1.1.2.4 Hildon . . . 3
    - 1.1.2.5 MAFW . . . 4
    - 1.1.2.6 OpenGL ES . . . 4
    - 1.1.2.7 Qt . . . 4
    - 1.1.2.8 Telepathy . . . 4
- 1.2 Requirements . . . 4
- 1.3 Conclusion . . . 5

2 Technologies **7**
- 2.1 Qt . . . 7
  - 2.1.1 History of Qt . . . 7
  - 2.1.2 Qt 4 Features . . . 8
- 2.2 HTML . . . 9

2.2.1 History of HTML 9
2.2.2 Basic structure 9
2.2.3 Cascading Style Sheets 10
2.3 QML 11
2.3.1 History of QML 11
2.3.2 Basic structure 12
2.4 JavaScript 13
2.4.1 History of JavaScript 13
2.4.2 Basic structure 14
2.4.3 JavaScript-HTML connection 14
2.4.4 JavaScript-QML connection 15

3 Comparison **17**
3.1 Application design 17
3.1.1 Main menu application 18
3.1.2 Contacts application 19
3.1.3 Media player application 20
3.2 Functional comparison 21
3.2.1 Animations 21
3.2.2 Code reuseability 25
3.2.3 Context Switching 26
3.2.4 Font support 28
3.2.5 Media playback 28
3.2.6 Object placement 29
3.2.7 Storage access 32
3.3 Performance comparison 34

4 Conclusion **37**

Glossary **39**

References **41**

A SubClassedApplication source code **43**
A.1 subclassedapplication.h 43
A.2 subclassedapplication.cpp 43

B Browser source code **45**
B.1 main.cpp 45
B.2 browser.h 45
B.3 browser.cpp 46

C Viewer source code 47
C.1 main.cpp 47
C.2 viewer.h 47
C.3 viewer.cpp 47

D Storage handler source code 49
D.1 storagehandler.h 49
D.2 storagehandler.cpp 49

E <div> table source code 51

F QML table layout 53

G Memory tests source code 55
G.1 Menu (V3) 55
G.1.1 HTML 55
G.1.2 QML 56
G.1.2.1 Menu.qml 56
G.1.2.2 Contents.qml 56
G.1.2.3 MenuItem.qml 56
G.2 Contacts (V3) 57
G.2.1 HTML 57
G.2.1.1 contacts.html 57
G.2.1.2 contactsCode.js 57
G.2.1.3 contactsStyle.css 58
G.2.2 QML 60
G.2.2.1 Contacts.qml 60
G.2.2.2 ContactAdder.qml 61
G.2.2.3 Button.qml 62
G.3 Player (V3) 62
G.3.1 HTML 62
G.3.1.1 player.html 62
G.3.1.2 playerCode.js 63
G.3.1.3 playerStyle.css 64
G.3.2 QML 66
G.3.2.1 Player.qml 66
G.3.2.2 playerCode.js 68

H Performance test source code 71
H.1 HTML 71
H.2 QML 71

List of Figures

1.1 Maemo OS200x Interfaces 2
1.2 Maemo 5 Interface 3
1.3 The Kano model 5

3.1 Main menu v. 1 18
3.2 Main menu v. 2 18
3.3 Main menu v. 3 18
3.4 Contacts v. 1 19
3.5 Contacts v. 3 19
3.6 Contacts v. 4 20
3.7 Contacts v. 5 20
3.8 Player v. 1 20
3.9 Player v. 2 21
3.10 Player v. 3 21
3.11 Simple table-style layout 29
3.12 QML layout 30
3.13 Memory performance graph 35
3.14 Aliasing in QML and QtWebKit 36

Introduction

This thesis is the result of my graduation project carried out at Nokia QDF (Qt Development Frameworks) in Oslo, Norway. During this graduation project I have compared QML and HTML technologies for developing mobile applications. The goal of the graduation project was to show that I can apply all knowledge I have gained during the previous three and a half years of the study 'Mediatechnology' at Utrecht University of Applied Sciences. Mediatechnology at Utrecht University of Applied Sciences is a four-year bachelor degree program containing courses regarding communication, design, software engineering and media systems. A student graduating from this program will be able to design and build multimedia applications.

QDF was founded as Quasar Technologies in 1994, later renamed to Trolltech and subsequently acquired by Nokia in 2008. The main product developed by QDF is the Qt toolkit. Qt is a cross-platform application development toolkit running on various operating systems providing an abstraction layer that allows developers to code once and deploy on several target platforms. Target platforms include Windows, Linux, Mac, Windows Mobile and Symbian.

This document is split out in several parts.
The first part contains information about the how and why of the project.
The second part is the core of this document, consisting of four chapters. The first chapter contains background information about popular types of mobile applications. Continuing I give information about the used technologies. The third chapter contains the actual comparison of the technologies. The fourth chapter is a conclusion of the results of the comparison.
The final part of this document contains the glossary, bibliography and appendices.

This document is written entirely in English since the graduation project was carried out abroad.

Problem definition

The sales figures for smartphones indicate that the smartphone market grew 12.3% in the third quarter of 2009 compared to the third quarter of 2008, even though the whole mobile segment only grew 0.1% between these periods [34]. This indicates that more features on phones is in great consumer demand.

Nokia has chosen Qt to become the main technology on Nokia smartphones and mobile computers [11]. Currently there are multiple development paths available in Qt tuned to develop feature-rich mobile applications, namely Qt C++, QML and QtWebKit. There is a strong push from within Nokia to use HTML and JavaScript technologies for mobile applications. These technologies have been combined in the Nokia WRT (Web Runtime). For high-performance applications Qt should be used.
Before being bought by Nokia, the people at QDF had started development of QML. The development of QML gives developers another option to easily develop mobile widgets. Therefore the problem tried to be solved in this thesis is:

Which of QML/JavaScript or HTML5/JavaScript is a better technology for developing mobile User Interface applications?

Herein QtWebKit is used for HTML5 development.

Assignment

As stated in the pages before, I need to find an answer to the question:

Which of QML/JavaScript or HTML5/JavaScript is a better technology for developing mobile User Interface applications?

QtWebKit will be used for the HTML5 content developed during this project. The answer to the central question should give Nokia a better understanding in determining a development path for mobile interfaces developed using Qt. I will have to compare the two technologies both feature-wise and performance-wise to be able to find an answer to the central question..

Sub questions

There are several sub questions which will help me find the answer to the problem:

- What kind of application is relevant on a mobile device with cellular capabilities?
- What is HTML?
- What is QML?
- What is JavaScript?
- What are the functional differences between HTML5 and QML?
- What are the performance differences between HTML5 and QML?

Objectives

At the end of the graduation project I want to have created a comparison report objectively describing the advantages and disadvantages giving the company a better understanding in choosing a development path for mobile interfaces based on Qt.

Products

The graduation project will yield:

- Developed comparison report.
- Multiple applications developed both in QML and HTML5. These applications should be as similarly featured as possible in both technologies.

Plan of approach

The project will be divided into several phases which each have a different goal. These phases are called the orientation phase, the comparison phase and the finalization phase.

- **Orientation phase (Feb 1 - Mar 24)**
  This phase is at the start of the graduation project and focuses on desk research, comparing what HTML, QML and JavaScript are, how they work, what their syntax is like and what their features and possibilities are. Firstly research is done on the mobile market space. *What are must-have applications and which features do these applications require?* The research on language features will then use information gathered during the first research to select focus areas.

- **Comparison phase (Mar 25 - May 24)**
  This phase starts with a short comparison of HTML and QML which should show obvious technology shortcomings. The next step is developing the applications. First a short design stage where the general application schematics will be laid out. The actual development will be done using the *Extreme programming* technique [44]. The final step in this phase is finishing the comparison report.

- **Finalization phase (May 25 - Jun 20)**
  The shutdown phase consists of a few weeks improving the applications and finishing the presentations to be done, both at the company and at the university.

1

Mobile

This chapter is about the background of the project. What kind of mobile platform will be used? What kind of applications are mandatory to have? What should be kept in mind? What can be the pain points? etc.

In this report 'mobile' refers to a cellphone running the Maemo mobile computer operating system. There is no official definition for what a smartphone is. This report uses the most commonly used definiton of smartphone, which is: "A smartphone is a large-screen, data-centric, handheld device designed to offer complete phone functions whilst simultaneously functioning as a personal digital assistant (PDA)." [15]

1.1 Maemo

Maemo is a software platform consisting of an operating system and an SDK (Software Development Kit) developed by Nokia, based on Debian GNU/Linux, developed in collaboration with several open-source projects like GNOME, Xorg and GStreamer [2]. One of the goals of the Maemo project is providing developers with a completely open development platform [4].

1.1.1 History of Maemo

The first public Maemo release was 'OS2005' for the Nokia 770 Internet Tablet in November 2005 [8]. This first release contained an Opera-based webbrowser, a few basic applications, utilites and games and can be considered as a first step in creating an open-source product [20].
In June 2006 a new Maemo version 'OS2006' was released, which contained a new kernel version, voice chat support, a finger-friendly on-screen keyboard and improved device performance. Due to API-breakage all existing applications had to be recompiled [21].
The next Maemo release was called 'OS2007', released bundled with the Nokia N800 in January 2007. This Maemo version contained an updated browser with Flash 7 support, podcast support and greatly improved performance due to the new hardware [22].

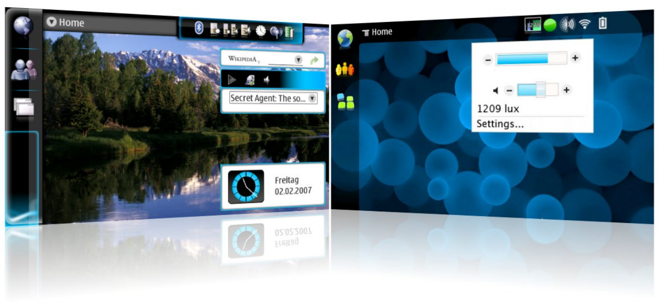

Figure 1.1: Maemo OS200x Interfaces - Screenshots of the OS2007 and OS2008 interface, displaying the home screens of both versions.

'OS2008' released in 2008 for the Nokia N810 was the next installment in the Maemo software platform series. This version exchanged the Opera browser for a Mozilla-based browser and gained Flash 9 support. Other notable improvements over the previous versions were a new home screen, official Skype support, a navigation application (due to the N810 gaining a GPS recevier which the older devices did not have) and UPnP/Windows network support [10]. There was a significant update to OS2008 which was codenamed 'Diablo'. Diablo contained an improved application management application which allowed for automatic software updates and contained a new email-client [41]. This was the last Maemo software platform with an 'OS200x' moniker.
Maemo 5 is the most recently released Maemo platform. It was released publicly at the 2009 Maemo Summit in Amsterdam, the Netherlands [45]. Key features of Maemo 5 are that it is the first Maemo version which runs on a device that has cellphone capabilities like calling, texting and mobile internet. Other key features of this platform version are a completely redesigned UI, official support for Qt, location sharing, improved multitasking support, integration with the Nokia Ovi application store and an improved web browser with full Flash 9.4 support [5]. Maemo 5 runs on the Nokia N900 Mobile Computer.

1.1.2 Maemo technologies

As mentioned before, the Maemo platform integrates several open-source technologies in one common user environment [28]. This subsection highlights some of the core technologies.

1.1.2.1 Clutter

The Clutter open-source software library is designed to make use of OpenGL hardware acceleration when rendering the user interface. There is support for behavioral animations, JSON scripts, media playback, physics and more. Clutter is developed by OpenedHand, which is currently a part of Intel [1].

1.1.2.2 GStreamer

GStreamer is used as the media handling library on Maemo. GStreamer itself is a library which allows a developer to 'create a graph of media handling components' [9]. This means that a developer can use several building blocks to create a multimedia application.
An extremely simple example of this is as follows:

```
gst-launch filesrc location=music.mp3 ! decodebin ! pulsesink
```

In this example the *gst-launch* utility is launched. This utility loads a file from location *music.mp3* and sends it to the *decodebin*. The decodebin decodes the file by automatically selecting the correct codec. The output of this decoding is then sent to the *pulsesink*. The pulsesink integrates with the Pulseaudio playback framework which will cause the audio to be sent to the selected output (either the speakers or the headphone plug) [40].

1.1.2.3 GTK+

GTK+ (The GIMP ToolKit) is a toolkit providing similar functionality to Qt. GTK+ was first developed for GIMP (GNU Image Manipulation Program). Over time GTK+ has developed to a toolkit which is used by a large number of applications and is the main component of the GNOME open-source desktop [39]. Known applications and solutions which are based on GTK+ include Mozilla FireFox and the OpenMoko phone interface. All Maemo interfaces up to Maemo 5 are written using GTK+.

1.1.2.4 Hildon

Hildon is a set of widgets based on GTK+. These widgets were developed to be more finger-friendly than normal GTK+ widgets. The interfaces of Maemo up to and including Maemo 5 were developed using Hildon.

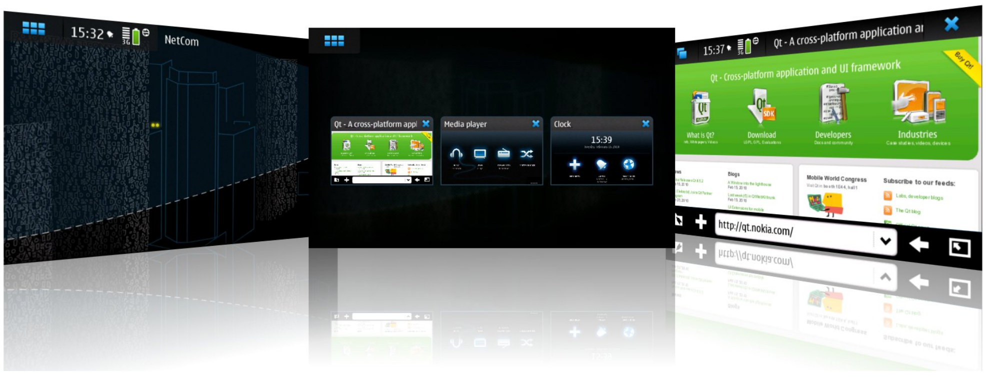
15:32
NetCom
Qt - A cross-platform appl
Media player
Clock
15:39
15:37
Qt - A cross-platform application a
Qt - Cross-platform application and UI framework
What is Qt?
Download
Developers
Industries
http://qt.nokia.com/

Figure 1.2: Maemo 5 Interface - Screenshots of the Maemo 5 interface, displaying the desktop, multi-tasking view and browser. These applications where created using Hildon.

1.1.2.5 MAFW

MAFW stands for Multimedia Application Framework and is a high-level plugin framework which eases integration of multimedia features like playlists and web services. Because MAFW is a plugin framework it is independent of underlying multimedia frameworks like GStreamer (see above). MAFW provides API for two kinds of plugins: Sources and Renderers. A Source plugin allows for content gathering, an example is a last.fm plugin. A Renderer plugin provides extended playback functionality (For example allowing a Maemo device to be a UPnP-server [42]).

1.1.2.6 OpenGL ES

OpenGL ES is an abbreviation for OpenGL for Embedded Systems. OpenGL ES is a subset of desktop OpenGL optimized for mobile device usage. OpenGL is a 3D Graphics API which enables hardware accelerated 3D rendering. There exist several versions of OpenGL ES. Version 1.x provides fixed-function pipeline API where all OpenGL coordinate transformations are done against a fixed matrix. Version 2.x replaces this fixed-function pipeline by a programmable-pipeline [6]. This allows developers to manipulate pixels and vertices after they have been put in the pipeline.

1.1.2.7 Qt

Qt is a cross-platform application development toolkit developed by QDF. Within Maemo, Qt will change from a community-supported toolkit to the officially supported platform toolkit with the change from Maemo 5 to Maemo 6 [28]. *More about Qt can be found in section 2.1.*

1.1.2.8 Telepathy

Telepathy is the library that provides instant messaging and VoIP possible on the Maemo platform. Telepathy is a plugin architecture focussed on real-time communication that lets applications share information about communcation availability. You could for instance create a messaging application that logs in to Windows Live Messenger. If you have a word processor that also communicates with the Telepathy framework then the word processor would ‘know’ about the availability of your Messenger contacts [7].

1.2 Requirements

Being a mobile phone there are certain requirements to be met. As indicated in fifure 1.3, [35] and [43] must-haves do not contribute to a positive consumer satisfaction per se, but they can cause dissatisfaction. Must-haves are certainly the most important part for mobile phone usage as indicated in [43]. These must-haves in a mobile phone mean that the user should have easy access to his contacts, messages, media and that he should be able to use simple applications such as a stopwatch and calculator.
Performance and exciting new features however have a prominent role in causing consumer

satisfaction. When the performance of the device is lower than expected this will lead to greatly deteriorated consumer satisfaction. Exciting new features cannot cause dissatisfaction, but they acount for a big amount in consumer satisfaction.

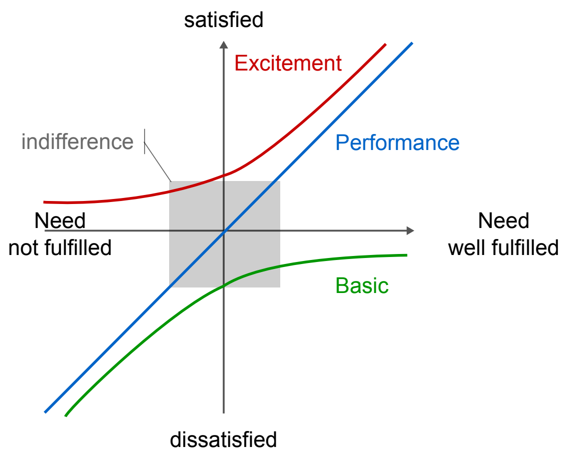

Figure 1.3: The Kano model - The figure shows the Kano model of consumer preferences. Adapted from [35]. The image visualizes the effect of features on consumer satisfaction and consumer needs.

1.3 Conclusion

Based on the platform capabilities and the requirements I will focus on simple must-have applications like a contacts, messaging and multimedia player application. The basic facilities for creating these applications are there. The research from [43] states that these basic tasks are the most important for a user group.
Another advantage of building these simple applications is that for the sake of this thesis there is no need of conceptualizing something new.

2

Technologies

2.1 Qt

This section contains background information about the Qt toolkit. Qt is a C++ application development toolkit written in C++ enabling developers to 'code less, create more, deploy everywhere' [24]. Qt contains several modules tailoring to developer needs. More about Qt modules can be found in subsection 2.1.2. During this project several parts of the Qt toolkit will be used.

2.1.1 History of Qt

In fifteen years time Qt has grown from a tookit used by a few people to a product used by millions of people daily [16].
The history of Qt started in 1988 when one of the two original Qt developers (Håvard Nord and Eirik Chambe-Eng) needed to make a GUI (Graphical User Interface) framework.
In 1990 Håvard and Eirik met to create a database application that needed to run on the Macintosh, Windows and UNIX operating systems. This application caused the two developers to think about making a full-fledged C++ GUI toolkit.
In 1993 the idea of the 'Signal-Slot mechanism' was devised and over the next two years the basis of Qt was laid. The name Qt was chosen because the letter Q looked nice in the font used by one of the developers and the t stands for toolkit.
In 1995 the two developers got their first customer and hired a third person. The coming ten months they would not get any other customer and relied on this first customer to provide income. Then finally in March 1996 the European Space Agency (ESA) became the second customer to use Qt.
In September of 1996 Qt version 1.0 was released. By now Trolltech (as the company was called at the time) had eight customers with a combined total of eighteen licenses. In 1996 the KDE project was also founded. The KDE project develops a Cross-Platform Desktop Environment. This desktop environment caused the Linux community to embrace Qt
Qt 2.0 was released in 1999 with a new Open-Source license called the Qt Public License (QPL). The following year Qt Embedded was released which led to a re-licensing of Qt

under the GNU General Public License (GPL).
In 2001 Qt 3.0 was released. Qt 3.0 was a huge step forward over Qt 2. Qt now contained over half a million lines of code and contained 42 new classes over Qt 2. Qt 3 did not provide support for embedded development.
In 2005 Qt 4 was launched which is the version that is still in development today. Qt 4 gained a new modular approach to application development, and added renewed support for embedded platforms. More about the Qt 4 modules in section 2.1.2.
Current well known solutions using Qt 4 are KDE, Google Earth, Autodesk Maya and Skype.

2.1.2 Qt 4 Features

Qt 4 has brought an enormous amount of improvements over Qt 3. Notably the split into several modules [26] [25]. Currently the following modules exist:

- QtCore - The main Qt module. This module contains all basic classes in Qt. Examples are QCoreApplication, QList and QString.
- QtDeclarative - The module containing classes for use with the Qt Declarative Markup Language (QML). The features provided by this module will be used in this project. QtDeclarative is a new module, first included in Qt 4.7.
- QtGui - The QtGui module contains classes for creating GUI applications. Classes for using the Graphicsview framework, QWidgets, toolbars, etc. can be found in this module.
- QtMultimedia - QtMultimedia is a module introduced with Qt 4.6. This module gives the developer low-level access to the multimedia hardware in a device.
- QtNetwork - QtNetwork makes it possible to create networked applications like web servers and clients.
- QtOpenGL - The module containing convenience API for developing OpenGL applications. There is a QGLWidget class that provides the developer with a surface to use for OpenGL rendering.
- QtSql - QtSql is a bridge for using database systems. There exist several database plugins for QtSql: MySQL, PostgreSQL, SQLite and others.
- QtSvg - Module that allows the loading of SVG vector graphics images.
- QtTest - The QtTest module is a convenience library to write unittests for a Qt application.
- QtWebKit - Module containing the Qt version of the WebKit web browser engine.
- QtXml - XML handling module. Makes parsing and editing XML documents possible.
- Phonon - High-level multimedia handling module. This module uses plugins called backends to play multimedia content. There are several backends including: DirectShow, Quicktime and GStreamer.
- Qt3Support - Convenience module for developers porting their applications from Qt 3 to Qt 4.

During the lifetime of Qt 4 several new important technologies have been added [27]. These include:

- The Graphicsview framework - This framework provides something similar to a painters' canvas where the developer can lay-out images and text.

- SVG export
- Phonon
- WebKit support
- Gestures support
- QML (Qt Declarative Markup Language)

2.2 HTML

HTML stands for HyperText Markup Language and is one of the technologies used in the comparison carried out during this project. Firstly the background of HTML will be explained, followed by important features and the usage of Cascading Style Sheets (CSS).

2.2.1 History of HTML

HTML is a continuation of Hypertext. A concept coined in 1965 by Ted Nelson [14]. There has been a lot of Hypertext development before that. In 1945 a proposal was made for a device that allowed a reader to go through 'linked' texts on microfilm: the Memex [18]. Over the next decades Hypertext would be developed and used further, the main goals being extending human intellect and creating linked documents [23].
Another important development of the time was the start of the ARPANET network. This would become the predecessor to the internet as we know it today [3]. This network would be used to send the first email messages.
In 1991 Tim Berners-Lee wrote a document specifying the first 20 HTML tags and contributed this document to a mailinglist [13][12]. This started the development of the HTML markup language. In 1995 HTML 2 was released which added official support for tables and image maps. HTML 3.2 was released in January 1997. HTML 3.2 tried to remove overlap between different proprietary HTML tags. HTML 4.0, released in December 1997, contained three variations (strict, transitional and frameset). In December 1999 HTML 4.01 was released with minor edits over HTML 4.0. HTML 4.01 is the most recent released HTML version.
HTML 5 will be the next HTML version. Currently HTML 5 is a working draft started in January 2008. HTML 5 is the HTML version used in this project.

2.2.2 Basic structure

HTML consists of tags. A tag defines an element, which can be a link, image, movie, etc. Each element starts with a start-tag and most elements end with an end-tag. Each tag starts with a less than-sign (<) and end with a greater-than sign (>). An end-tag differs from a start-tag in that it has a forward slash (/) after the less-than sign. Some tags can be closed immediately, like the <br /> tag which is used for a linebreak. Here the forward slash is at the end of the tag to signify that the tag is being closed immediately.
Let's start with a simple example of HTML code.

```
<b>Hello</b>
```

The <b>-tag is used to define a section of text that should be printed in a bold font. Therefore **Hello** would be displayed.

Each HTML document has a standard structure which can be extended.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Document title</title>
  </head>
  <body>
    <p>Contents of the first paragraph.</p>
  </body>
</html>
```

The <!DOCTYPE>-tag is used to specify the schema of the document. In this case the document identifies itself as a HTML document.
The <html> and </html> tags are used to specify which part of the document is HTML code. Text outside of the <html> and </html> is treated as plain-text.
In the head section of the document the title, meta information and external script sources should be specified. In this example the document uses UTF-8 encoding and it's title is 'Document title'. This title is displayed as window or tab title in the browser.
The body of the document contains all data that should be presented to the user of the webpage. Paragraphs within the body are separated by <p> and </p>-tags.

An important aspect of Hypertext is the linking of external sources. In HTML 5 this can be done with several tags, depending on the content:

- <a> - for linking to another (part of a) document.
- <audio> - for audio files.
- <embed> - for embedding content that should be loaded through a plug-in, like Flash content or a Java applet.
- <img> - for images.
- <link> - for linking external page resources. This tag is mostly used for linking style sheets.
- <video> - for video files.

2.2.3 Cascading Style Sheets

Cascading Style Sheets (CSS) are used to specify styling hints for markup languages like HTML [17]. CSS was designed to separate document content (which is in the HTML file) from document presentation. This separation makes it easier to read the HTML-file and makes it possible for separate pages to use the same styling without repeating the styling information.

A Cascading Style Sheet has to be added to HTML files using the <link>-tag.

```
<link href="stylesheet.css" rel="stylesheet" type="text/css" />
```

The href property specifies the location of the CSS file. The rel-property specifies the usage of this link, which should always be set to 'stylesheet' for loading a stylesheet. Similarly the type-property should always be set to the 'text/css'-mime-type.

CSS consists of selectors, classes and ID's [29]. Any HTML element is a selector by default. In a CSS-file the names of the standard HTML elements are used without a prefix. A simple way to get the page background to be red would be by setting the background-color property of the body element to red. In CSS this looks like this:

```
body { background-color: red }
```

ID's and classes are special selectors. An ID is used to specify a unique object within the HTML DOM tree, for instance a sidebar. ID's are prefixed by a hash (#) character.

```
#sidebar { font-family: sans-serif }
```

In this example the font of the element with 'sidebar' as its id property will be sans-serif. Classes are used for common elements, for example a class called 'comment' would be applied to all comments in the page. Class selectors are prefixed by a dot (.) character.

```
.comment { font-size: small }
```

In this example the font size of all elements with 'comment' as their class property will be small.

2.3 QML

QML stands for Qt declarative Markup Language. QML is the second technology used in the comparison carried out during this project. Firstly there is a subsection about the history of QML followed by a basic technology overview.

2.3.1 History of QML

Development of the framework behind QML has started somewhere near the end of 2007, start of 2008 [31]. At that time there was a demand for highly animated GUIs which could not be done in Qt at the time. This new project was called “Qt Kinetic” and it's goal was to enable developers to easily create highly animated GUIs without the need to manage complex data structures which are inherent to complex animated GUIs.
At this time QML was still a language similar to HTML and XML in that it used XML to describe objects [31]. Over the next few months the Qt Animation framework was implemented along with a state machine [30][32].
In May 2009 QML had changed to a different syntax [38], the syntax now looked less like XML and more like JSON (JavaScript Object Notation). At the same time the Anchor Layout was also in development [37]. In January 2010 support for QML was integrated in the Qt IDE.

2.3.2 Basic structure

QML is an interpreted language where a runtime engine parses the content of the QML files, which in turn generates Qt C++ QML elements. Several standard QML elements are shipped with Qt, like the Rectangle, TextEdit, multimedia elements and elements to handle animations.

As stated above QML has a syntax similar to JSON. QML code to create a rectangle that which 100 by 100 pixels big and placed at the 0,0 position one would look as follows:

```
Rectangle
{
    width: 100
    height: 100
}
```

The name of a QML element always starts with a capital letter. The contents of the element are enclosed in curly braces. Note the usage of the colon in the property definition. This colon indicates the creation of a property binding rather than a property declaration. What this means is best demonstrated using an example.

```
Rectangle
{
    id: firstRectangle
    width: 100
}

Rectangle
{
    id: secondRectangle
    width: firstRectangle.width
}
```

If the width of firstRectangle was now to change the width of secondRectangle would also change automatically. This binding of properties is one of QML's most important features.

Custom C++ QML elements can be developed by using a macro and a template function.

```
#define QML_DECLARE_TYPE(T)
template<typename T>
int qmlRegisterType(const char *uri, int versionMajor,
                    int versionMinor, const char *qmlName)
```

Type *T* is the class name of the class that should be available from QML. The qmlRegisterType template adds the QML type to the pool of available elements. This new custom element can now be used from a QML file by using the name defined as *qmlName* in the qmlRegisterType(...) macro.
The prefered method to develop custom QML elements through C++ is by using a QDeclarativeExtensionPlugin. This plugin can be loaded by the 'qml' runtime application shipped with Qt and allows the elements which are defined within this plugin to be used in QML applications. A minimal QDeclarativeExtensionPlugin class looks as follows:

```
#include <QtDeclarative>

QML_DECLARE_TYPE(CustomElement);

class CustomElementExtensionPlugin : public QDeclarativeExtensionPlugin
{
    Q_OBJECT

    public:
        void registerTypes(const char *uri)
        {
            qmlRegisterType<CustomElement>(uri, 0, 1, "CustomElement");
        }
};

Q_EXPORT_PLUGIN(CustomElementExtensionPlugin);
```

This class registers the type *CustomElement* in the QDeclarativeEngine who calls this plugin method.
QML C++ plugins are not loaded automatically. The plugin has to be specified in a 'qmldir' file. a 'qmldir' file is a file containing all external resources which the QML runtime should load. If the qmldir file is located in the same directory as the QML file which is loaded an import statement is not needed within the loaded QML file. To be able to use our plugin we should add a line to the qmldir file:

```
plugin customelementextensionplugin
```

This will cause our extension to be loaded and automatically make the QML types defined using the qmlRegisterType(...) macro available.

2.4 JavaScript

JavaScript is the scripting technology used in both HTML and QML. The first two subsections deal with the history and structure of JavaScript. Continuing there are two subsections describing the way JavaScript interacts with HTML and QML respectively.

2.4.1 History of JavaScript

At the start of the common internet webpages were completely static. There was a need to be able to interact with a webpage. To be able to handle this interaction immediately there would need to be an embedded programming language within the browser. To this end JavaScript was developed [19].

JavaScript was developed by Netscape in 1995. At first it was called Mocha, later renamed to LiveScript and, due to a licensing deal where Netscape would integrate Java in its web browser, JavaScript [36].

Over the next years JavaScript has become the dominant web scripting technology gaining even greater dominance after the creation of the "AJAX" combination of dynamic web page creation technologies and the evolving of Web 2.0.

2.4.2 Basic structure

JavaScript was developed to be used by non-programmers. Because of this it was decided that some features used in multiple other languages would not become available in JavaScript to be able to decrease the learning curve.

JavaScript consists of methods called *functions* which can be called from functions. A simple function that adds two variables and returns the result would look like this:

```
function adder(one, two)
{
    return one + two;
}
```

Notice there is no variable *type* specified for the variables. All variables in JavaScript are casted automatically. The JavaScript engine verifies the validity of an operation and returns ‘Not a Number’ (NaN) in case of a numeric operation failing and ‘null’ in other cases.

JavaScript does not contain explicit classes. The developer can define JavaScript functions which have other functions as prototypes, making them essentially member functions. For instance:

```
function addContact(firstName, lastName, phoneNumber)
{
    this.firstName = firstName;
    this.lastName = lastName;
    this.phoneNumber = phoneNumber;
}

function showName()
{
    alert("Name of contact: " + this.firstName + " " + this.lastName);
}

addContact.prototype.showName = showName;
```

In this example *showName()* is a function of the *addContact()* class. When a variable is now defined to be of the addContact type you can call variable.showName():

```
function display()
{
    var contact = new addContact("John", "Doe", "+123456789");
    contact.showName();
}
```

In this last example an alert box will pop-up with the content: “Name of contact: John Doe”.

2.4.3 JavaScript-HTML connection

JavaScript objects can be instantiated in an HTML page in two ways. Either through external files in the head of the document:

```
<head>
    <script type="text/javascript" src="pagescript.js"></script>
</head>
```

Or by embedding the JavaScript content within the body of the document:

```
<body>
<script type="text/javascript">
document.write("JavaScript text");
</script>
</body>
```

The actual modification of the HTML file is done through the internal representation of HTML objects, which is called the DOM (Document Object Model) tree. In the above example the text "JavaScript text" is inserted at the position of the codeblock.

JavaScript can also be used to insert content at arbitrary positions. To do this JavaScript contains the document.getElementById method which uses the same ID's as CSS (see section 2.2.3).

```
<script type="text/javascript">
function notEmpty(){
    var myTextField = document.getElementById('myText');
    if(myTextField.value != "")
    {
        alert(myTextField.value);
    }
    else
    {
        alert("The textfield is still empty.");
    }
}
</script>
<input type='text' id='myText' />
<input type='button' onclick='notEmpty()' value='Form Checker' />
```

In this example the ID of the text field is 'myText' which is assigned to the *myTextField* variable in JavaScript. When the user clicks on the button JavaScript checks the contents of this text field and shows a pop-up containing either the text in the text field or the string "The textfield is still empty.".

2.4.4 JavaScript-QML connection

JavaScript is a core component of QML. All operations written within a QML file are basically JavaScript operations. Let's demonstrate this behaviour:

```
width: parent.width - 5
```

In this example the width of the current QML element is the width of its parent decreased by five. The change of the width property is calculated by evaluating the expression using the JavaScript parser. In this example the property binding makes sure that the width of the current item is always five pixels smaller than its parent's with (see subsection 2.3.2).

In QML the developer can also declare and use JavaScript functions. functions defined within a QML element can be used to add methods to this specific QML element, these functions are not available to other elements:

```
Rectangle
{
    function adder(one, two)
    {
        return one + two;
    }

    MouseRegion
    {
        anchors.fill: parent
        onClicked: print(adder(1, 2))
    }
}
```

The developer can also choose to have the JavaScript code in an external file, which makes all functions in this JavaScript file available to all QML elements in the QML file which imports the JavaScript file.
Using an external file with JavaScript code requires this JavaScript file to be imported:

```
import "code.js" as Code
```

The keyword following the 'as' identifier is now to be used to access functions within this JavaScript file. The following example behaves exactly the same as the above example, except for that the code is now in an external file.

```
import "code.js" as Code

Rectangle
{
    MouseRegion
    {
        anchors.fill: parent
        onClicked: print(Code.adder(1, 2))
    }
}
```

QML exposes all functions in the JavaScript global object and adds some extra functionality to this global object. Added functionality includes generating MD5 hashes, converting date and times and the modification of colors by applying a tint to the color. The more important added functionalities however are the addition of a mechanism to dynamically generate QML elements, access a local SQLite database and the addition of methods to query external data using XmlHttpRequest.

3

Comparison

3.1 Application design

In section 1.3 the conclusion was: Users expect several basic applications to be available on every phone. This is the reason for the decision to only develop simple applications which everybody expects on a mobile phone. These applications should however still contain most use cases a mobile application developer would encounter:

- Navigation between application phases - A game developer would want to go from the main menu to the game.
- Loading and saving content - This would be usable for a Twitter application which stores tweets locally.
- Scrolling
- Animations
- Media playback
- Orientation changes - Most mobile phones use a rectangular screen and widescreen media content. Users would want to display their media content as big as possible.

Three applications will be developed:

- A main menu to get to the other applications. This application will be used to test switching between application phases.
- A contacts application where it should be possible to add and remove contacts. This application will be used to test scrolling and storage functionality.
- A media player which should be able to play back an audio file. This application is used to test animations, media playback and orientation changes.

Each application will have several versions which will add new features. The usage of versions will make it easier to compare feature differences, differences in amounts of code needed and differences in performance. An added advantage is decreasing the risk of failure. If certain requirements are not met for a specific version an effort can be made to develop a later version.

The applications are compared using the ‘qml’-runtime application for QML and a simple

QtWebKit browser (sourcecode in appendices A and B) for HTML. All applications will be developed to use the least amount of lines of code while retaining all necessary features.

3.1.1 Main menu application

The main menu application is the simplest application of the three. The main goal of the main menu application is to get to the other applications. This application is used to test the ability to easily load other applications within the same environment.

The main menu application has the lowest priority. Version 1 is the must-have version to be usable. All other versions have 'would like to have' priority.

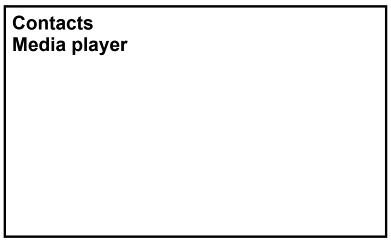

Figure 3.1: Main menu v. 1 - Version 1 of the main menu contains two clickable items which are plain text.

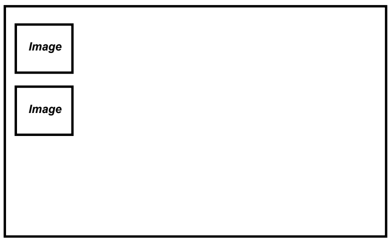

Figure 3.2: Main menu v. 2 - Version 2 replaces the text from version 1 by images.

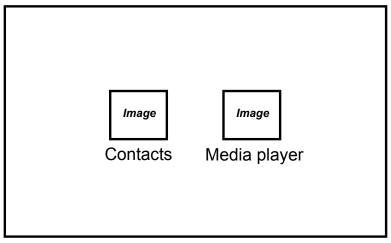

Figure 3.3: Main menu v. 3 - Version 3 moves the images from version 2 to the center and adds the name of the application.

Version 4 of the main menu application will look exactly the same as version 3, except that the menu content is now dynamically loaded and not explicitly typed anymore. This means that the content of the folder containing the applications should be automatically read and image buttons should be placed at their respective places automatically.

3.1.2 Contacts application

The contacts application is the most data intensive application. The main goal of the contacts application is allowing for data manipulation. The contacts application is used to test scrolling and storage functionalities.

The contacts application has five versions. Of which version 1, 2, 3 and 4 have 'must-have' priority. Version 5 would be nice to have, but is not mandatory.

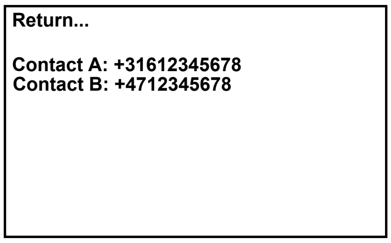

Figure 3.4: Contacts v. 1 - Version 1 of the contacts application contains a hard-coded text list of contacts and a text-link to return to the menu.

Version 2 of the contacts application looks the same as version 1. The hard-coded list will be replaced by a static database. This database should be cross-platform and preferably the same database should be used by both the QtWebKit and QML application.

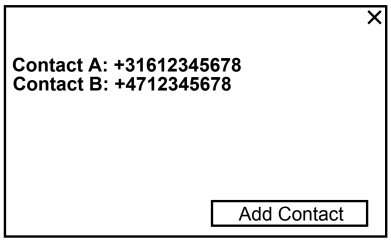

Figure 3.5: Contacts v. 3 - Version 3 of the contacts application allows for adding contacts to the database and replaces the return-text by an icon.

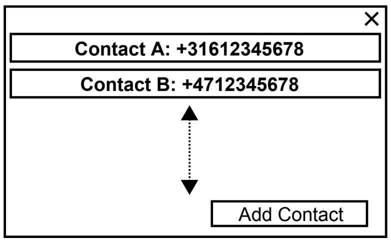

Figure 3.6: Contacts v. 4 - Version 4 of the contacts application places the contacts in boxes and allows for scrolling the content.

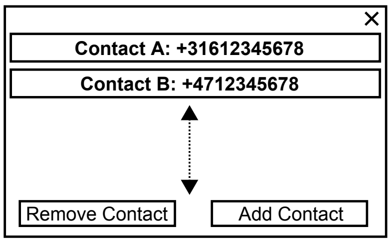

Figure 3.7: Contacts v. 5 - Version 5 of the contacts application adds a 'remove contact' button. To be able to remove a contact the user should also be able to select the contact to delete.

3.1.3 Media player application

The media player application focuses on playback of local files and device orientation changes. This application is used to compare screen orientation capabilities, media playback and animation functionality and performance.

All three versions of this application have to be developed to compare all of these functionalities.

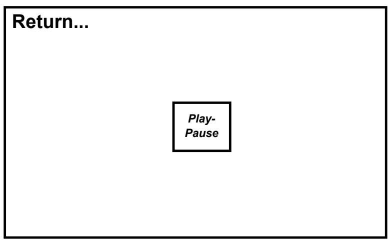

Figure 3.8: Player v. 1 - Version 1 of the media player application only contains a button which changes from play to pause and causes an audio file to play or pause.

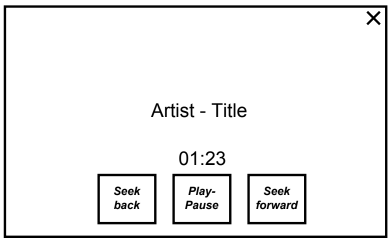

Figure 3.9: Player v. 2 - Version 2 of the media player application adds capabilites for navigating through the audio file, displays meta-information about the current file and shows the current time.

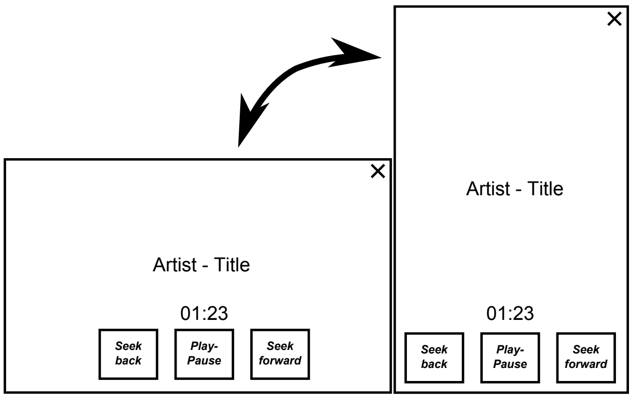

Figure 3.10: Player v. 3 - Version 3 of the media player application adds support for arbitrary screen orientation changes (Landscape to Portrait). When the screen orientation changes all objects within the visible area have to animate to their new positions.

3.2 Functional comparison

3.2.1 Animations

QML provides three different ways to specify animations. The first method specifies an animation following an action. This method uses elements like PropertyAnimation and ParallelAnimation. To animate a rectangle moving right 50 pixels when it is clicked the following could be used:

```
Rectangle
{
    color: "black"
    height: 150
    id: rectangle
    width: 150

    PropertyAnimation
```

```
        {
            duration: 200
            id: animation
            property: "x"
            target: rectangle
            to: rectangle.x + 50
        }

        MouseArea
        {
            anchors.fill: rectangle
            onClicked:
            {
                animation.start()
            }
        }
    }
```

The animation that is used here is exactly the same every time. Therefore a default animation can also be specified. The Behavior element can be used to specify these default animations for property changes. Using the Behaviour element is the second method for QML animations. To automatically animate the rectangle moving to its new position when the x-property changes using a behavior looks like this:

```
    Rectangle
    {
        color: "black"
        height: 150
        id: rectangle
        width: 150
        Behavior on x
        {
            NumberAnimation
            {
                easing.type: "Linear"
                duration: 200
            }
        }

        MouseArea
        {
            anchors.fill: rectangle
            onClicked:
            {
                rectangle.x = rectangle.x + 50;
            }
        }
    }
```

In this example the *Linear* easing type is specified. QML contains more than 40 easing types by default, ranging from the standard linear and parabolic functions to elastic and bouncy curves like InElastic and OutBounce.
The third and final method for animations in QML is the use of states and transitions. A QML State element describes the properties for elements when the application is in a specific state. States can be used to specify the look and feel of an application between portrait and landscape orientations. To have an animated switch between states the Transition element

is used. The example where the rectangle is moved when it is clicked can also be developed using states. A big difference however is that when using a behaviour or property animation the animation is done again when the rectangle is clicked again. When using states the object can only move once to go to the new state.

```
Rectangle
{
    color: "black"
    height: 150
    id: rectangle
    width: 150

    states: [
         State {
             name: "moved"
             PropertyChanges {
                 target: rectangle
                 x: 50
             }
         }
     ]

    transitions: [
         Transition {
             NumberAnimation { properties: "x"; duration: 200 }
         }
     ]

    MouseArea
    {
        anchors.fill: rectangle
        onClicked:
        {
            rectangle.state = "moved"
        }
    }
}
```

Animations can also be used in sequence or parallel and a combination of those, for instance to run parallel animations in sequence.

QtWebKit supports two types of animations: transitions and keyframe animations. Animations are not an official part of the HTML5 specification yet, this means that for now WebKit specific commands need to be used. All animations in WebKit are style options described through CSS3.

CSS3 transitions specify how an HTML object will change from its previous state to the new state. These transitions are applied for every modification that is done, no matter how big the change in the animated property is. An example to fade out a div using an opacity transition:

```
<body>
<style type="text/css">
    div
    {
        border-style: solid;
        border-width: 3px;
```

```
            opacity: 1;
            -webkit-transition: opacity 1s linear;
        }
        div:hover
        {
            opacity: 0;
        }
    </style>
    <div>&nbsp;</div>
    </body>
```

In this example the opacity changes from 1 to 0 in one second, where the duration is not related to the change of the property. This means that if the developer changes the new opacity to be 0.5 instead of 1, the transition will still take one second, effectively doubling the animation time. This transition uses the *linear* transition type, WebKit CSS3 contains six transition types: ease, linear, ease-in, ease-out, ease-in-out and cubic-bezier. Cubic-bezier allows the developer to define two points that will be interpolated to a curve that should be used for the animation.

The keyframe animation type in WebKit is used to describe fixed animation. A keyframe animation can have *from* and *to* properties for a plain start and ending animation, or percentages to describe the sub-steps within the animation. When the percentage approach is used it is possible to create a sequential animation:

```
    <body>
    <style type="text/css">
        @-webkit-keyframes moveandfade
        {
            0%
            {
                margin-left: 0;
                opacity: 1;
            }
            33%
            {
                margin-left: 15px;
                opacity: 0.5;
            }
            67%
            {
                margin-left: 70px;
                opacity: 1;
            }
            100%
            {
                margin-left: 100px;
                opacity: 0.5;
            }
        }
        div
        {
            border-style: solid;
            border-width: 3px;
            opacity: 1;
        }
        div:hover
        {
            -webkit-animation: 'moveandfade' 2s;
        }
    </style>
    <div style="width: 100px">&nbsp;</div>
    </body>
```

In this example the div will move from 0 to 100 on the x-axis while fading in and out.

3.2.2 Code reuseability

Code reusability in HTML is achieved through JavaScript and CSS. The style and code are split out between a CSS and a JavaScript file. Let's demonstrate this by developing a button which calls a function upon being clicked and has its text centered horizontally and vertically with top and bottom margins which are 3 pixels big and left and right margins which are 10 pixels big. To achieve this the CSS file specifies the margins around the text:

```
    .button
    {
        padding: 3px 10px; /* top and bottom 3px, left and right 10px */
    }
```

And the JavaScript file combines this styling information with an onclick handler and the actual text.

```
    function Button(text, functionToCall)
    {
        document.write("<div class=\'button\' onclick='" +
                        functionToCall + "'>" + text + "</div>");
    }
```

Note that the JavaScript code uses a class from the CSS file, but does not actually 'know' the contents of this file. The HTML page itself should include both the CSS and JavaScript code and handle accordingly. This is made visible in the following example where the JavaScript Button function is used to insert a *div* which has 'button' set as its class property.

```
    <head>
        <link href="button.css" rel="stylesheet" type="text/css"></link>
        <script src="button.js" type="text/javascript"></script>
    </head>
    <body>
        <script type="text/javascript">
            Button('Test', 'otherFunction()');
        </script>
    </body>
```

All HTML pages that want to use the JavaScript Button need to duplicate the two lines contained within the head section as displayed in listing.

QML is an object-oriented markup language which makes it easy to reuse existing code.

Each code block that should be reusable should be defined as a QML-Component.

An example using a Button component follows. In this Button component QML bindings are used to simplify keeping track of the size of the component. Whenever the 'text' property on the Button component is set it is automatically drawn in the buttonText Text component which automatically resizes the container Rectangle. The look and feel of this QML button are the same as in the HTML example above.

```
Rectangle
{
    id: container

    signal clicked
    property string text

    width: buttonText.width + 20
    height: buttonText.height + 6

    MouseArea
    {
        anchors.fill: parent
        onClicked: container.clicked()
    }

    Text
    {
        anchors.centerIn: container
        id: buttonText
        text: container.text
    }
}
```

This component can now be used when the component is defined both in the 'qmldir' file in the directory containing the component definition and in an import statement in the QML-file using the component, where the current directory is automatically imported.

```
Button
{
    text: "Button"
    onClicked:
    {
        otherFunction()
    }
}
```

3.2.3 Context Switching

The main menu application is used to switch between contexts. The user will leave the menu context and go to either the contacts of media player contexts. This function is especially important in games. In an ordinary game there is a loading screen before entering the game. At the moment the loading screen disappears there is a context switch from the loading context to the gaming context.

In QML the Loader-element is used to switch contexts. The Loader-element allows the developer to load QML code from other files. Loading code through the Loader-element however appends the content of the loaded data to the current file. It does not replace the current context. The easiest way to overcome this problem is by specifying a top-level QML-file which only contains a loader and putting each context in a separate file. To achieve this behavior the developer has to do the following:

```
//Menu.qml
Loader
{
    source: "Contents.qml"
}

//Contents.qml
Item
{
    Text
    {
        text: "Other file."

        MouseArea
        {
            anchors.fill: parent
            onClicked:
            {
                root.source = "Other.qml"
            }
        }
    }
}
```

The context of the application will now be changed to “Other.qml” when the user clicks anywhere in the application.
QML does not provide a built-in solution to return to previous contexts. The developer has to implement this functionality.

Context switching is the basis of Hypertext, which should mean that it is particularly easy to switch contexts in HTML. Context switching in HTML takes one line of code:

```
<a href="other.html">Other file.</a>
```

When this link is clicked the previous context is completely replaced and cannot be referenced anymore. Extra content can be appended to the link to be able to cache some data from the previous context.
All links in an anchor tag are hardcoded, to create dynamic links the developer has to use JavaScript to replace the content of the *href* property of the link. In HTML it is difficult to return to previous contexts. One way to circumvent this is by using the built-in JavaScript *history.go()* function. There is no guarantee however that the user will end up at the correct page.
Because the previous context is deleted and a new one is created it is hard to animate the changes between contexts. One would then have to animate the new context directly while it is being loaded. HTML does not specify exactly when content should be displayed, which means that unwanted behavior can occur.

3.2.4 Font support

QML and QtWebKit both automatically support the fonts which are installed on the users system. Both technologies also support loading additional fonts.

QtWebKit supports loading additional fonts through CSS. The @font-face CSS3 rule has to be used to load these custom fonts.

```
@font-face
{
    font-family: 'Nokia Sans';
    src: url('Nokia_Sans.ttf');
}
```

The Nokia_Sans.ttf font can now be used as any font in CSS. The developer has to use the name used in the font-family property of the @font-face rule.

```
h1
{
    font-family: "Nokia Sans";
}
```

Loading additional fonts in QML is possible through the FontLoader element.

```
FontLoader
{
    id: nokiaFont
    source: "Nokia_Sans.ttf"
}

FontLoader
{
    id: webFont
    source: "http://www.princexml.com/fonts/steffmann/Starburst.ttf"
}
```

The FontLoader element loads the font from a source, which can be either a local file or a network resource which will be downloaded automatically.
The special fonts can now be used by setting the font-family of a QML element containing text to the name of the loaded font.

```
Text
{
    font-family: nokiaFont.name /* or "Nokia Sans" */
}
```

3.2.5 Media playback

Media playback functionality between QtWebKit and QML is almost the same even though QtWebKit uses Phonon and QML uses the QtMobility frameworks to play back media content. These frameworks give access to all codecs which are installed on the user system.

HTML5 allows a developer to use the browser default media controls by specifying *controls="true"* on an audio or video element. Controls can also be developed using JavaScript and CSS.

QML does not contain default controls for multimedia elements. QML does add a SoundEffect element, which is a low-level implementation usable for short sounds. The SoundEffect element is based on QSound, which depending on the platform, enables functionality to use only WAVE files or also give functionality to use other codecs.

3.2.6 Object placement

Object placement is about getting the content at the position where the designer wants the content to be. Lets explain how all of this works by using a simple example layout which will be used in this subsection:

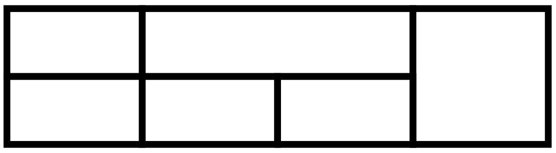

Figure 3.11: Simple table-style layout - A simple example of what an application layout can look like.

This layout contains two rows and elements with are double-height an double-width.

QML provides three different layouting options: relative to parent, positioners and anchor layout. All QML elements are automatically layout relative to their parents. Lets take the following example:

```
Rectangle
{
    color: "red"
    id: redRectangle
    height: 100
    width: 100
    x: 100
    y: 0

    Rectangle
    {
        color: "green"
        height: 100
        width: 100
        x: 100
        y: 0
    }
}

Rectangle
{
    color: "blue"
    height: 100
    width: 100
    x: 200
    y: redRectangle.height + redRectangle.y
}
```

This example will be rendered as seen in figure 3.12. Notice how the green rectangle is positioned at (200,0). This is because of the relative layouting to its parent (the red rectangle). The blue rectangle is position vertically using QML property bindings. These bindings will make sure that when either or both the height and y-position of the red rectangle change, the position of the blue rectangle will be updated automatically.

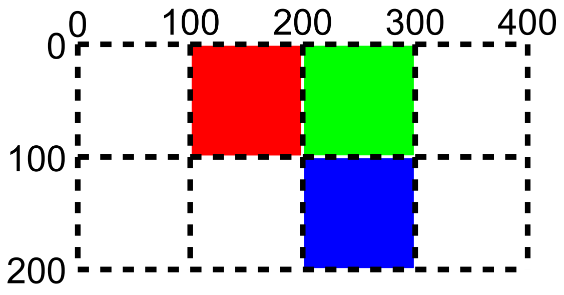

Figure 3.12: QML layout - An example of a QML layout. This will be rendered without the dashed lines

Positioners are used to layout QML Elements in columns, rows, a grid or a 'flow'. Columns and rows speak for themselves, the grid places elements in a grid. This grid does not however contain functionality which allows elements to have a width or height that span multiple rows or columns. The 'Flow' positioner aligns elements side by side while wrapping as necessary. The layout displayed in 3.12 cannot be developed using positioners.

The QML anchorlayout can also be used to develop the layout displayed in figure 3.12:

```
Rectangle
{
    color: "red"
    id: redRectangle
    height: 100
    width: 100
    x: 100
    y: 0
}

Rectangle
{
    anchors.left: redRectangle.right
    anchors.verticalCenter: redRectangle.verticalCenter
    color: "green"
    height: 100
    width: 100
}

Rectangle
{
    anchors.left: redRectangle.right
    anchors.top: redRectangle.bottom
    color: "blue"
    height: 100
    width: 100
}
```

The left of the green rectangle will always be aligned to the right of the red rectangle and the green rectangle will always be vertically centered with the red rectangle, so if the height of the red rectangle increases the green rectangle will move down. The blue rectangle is also aligned to the right of the red rectangle and to the bottom of the red rectangle. These concepts can also be applied to develop the table layout in figure 3.11. The QML code for this layout which is completely scalable can be found in appendix F. Note that half of the border width is on the inside of the element and half on the outside, therefore the whole canvas is 602x202 pixels to display a 600x200 table with 2 pixel borders.

HTML provides three different layouting options: layout using tables, layout using divs and the HTML5 canvas. Tables and divs are the original HTML way of layouting items, where text is the content that should be visible best. When the text does not fit within the table cells, or div, the rendering engine will automatically resize the table cell or div to accommodate the extra content. The HTML5 canvas on the other hand is completely pixel oriented and does not resize elements automatically. The HTML <table> element is used to create tables. Within a table there are table rows and table cells which use the <tr> and<td> elements respectively. Table cells can encompass multiple rows or columns using the *rowspan* and *colspan* properties. The HTML code to create the table as displayed in figure 3.11 looks as follows:

```html
<style type="text/css">
    td
    {
        border-style: solid;
        border-width: 1px;
    }
</style>
<table width="600px" height="200px">
    <tr>
        <td>&nbsp;</td>
        <td colspan="2">&nbsp;</td>
        <td rowspan="2">&nbsp;</td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
</table>
```

This table can also be created using only <div> elements. The code for the table created using <div> elements can be found in appendix E. The thing that is immediately clear when looking at the source code for the div-table is the amount of CSS needed to create the table layout. <div> elements are only invisible containers that can be modified, this modification has to be done through CSS. An interesting point is the need for an invisible cell and extra subdivision cells to create double-height and double-width cells. Since the table and div element both automatically resize according to their content, there has to be at least some content in the elements. *&nbsp;* prints a non-breakable whitespace. The elements will look empty, but there is content. The div and table elements can only use percentage and pixel widths and heights. It is impossible to specify the width of a div relative to the size of another div using plain HTML/CSS. Relative sizes can be used when the DOM-tree is modified using JavaScript.

The final possibility to create the table layout is using the new HTML5 <canvas> element. Like the div-element the canvas is only a container. A big difference between the div and the canvas however is the fact that the canvas has to be filled through JavaScript and is pixel-based. As such the canvas does not contain objects that can be filled. The text is completely separate from the drawing elements. The canvas element will also not resize to fit its contents automatically. Extra content will be clipped off.

```
<head>
    <script type="text/javascript">
        function fillCanvas()
        {
            var elem = document.getElementById('myCanvas');
            var context = elem.getContext('2d');

            context.lineWidth = 2;
            context.strokeRect(1, 1, 150, 100);
            context.strokeRect(151, 1, 300, 100);
            context.strokeRect(451, 1, 150, 200);
            context.strokeRect(1, 101, 150, 100);
            context.strokeRect(151, 101, 150, 100);
            context.strokeRect(301, 101, 150, 100);
        }

        window.onload = fillCanvas;
    </script>
</head>
<body>
    <canvas id="myCanvas" width="602" height="202"></canvas>
</body>
```

The above code is used to create the table layout using the 2d-version canvas element, the getContext() method is used to select either a 2d or a 3d context.. Notice how the topleft rectangle has to start on (1,1). This is the result of the automatic clipping of the canvas element. All strokes that are painted have half of their stroke on the inside and the other half on the outside. Since the selected lineWidth is 2, there should be a spacing of 1 pixel to not clip off half of the line. To make sure there is also no clipping at the bottom right corner the width of the canvas has to be at least 602x202 to accomodate the table which is 600x200.

3.2.7 Storage access

Both QML and QtWebKit can use the HTML5 LocalStorage API to store data. This LocalStorage API is however only to be used as a temporary cache, since the LocalStorage API does not offer any guarantees about data availability [33].

For persistent data storage both QML and QtWebKit support Qt C++-objects and an HTTP REST application service. When using Qt C++-objects the developer extends the runtime in which the application runs. In the case of QtWebKit this can be seen as a browser-extension. When using the QML it is a plugin which is plugged into the QML runtime environment (see subsection 2.3.2).

A Qt C++ object which is to be used by QML and/or QtWebKit has to be a QObject derived class. Within this class, properties defined using the Q_PROPERTY() macro and member functions defined as 'public slots' will be available to the runtime as properties and functions respectively. In appendix D, the sourcecode used to give storage access to both QtWebKit and QML can be found. The code used is almost the same except for the difference in linebreak characters between QML and HTML.

To give a webpage access to the methods defined in the Qt C++ browser extension a reference to its class has to be passed to the webpage global JavaScript object. This is done using the QWebFrame::addToJavaScriptWindowObject(const QString &name, QObject *object) method. Once this method has been called, an object with the name specified in *&name* is available on the webpage.
Since for every webpage the global JavaScript object is cleared, the addToJavaScriptWindowObject(...) should be called every time a new page is loaded. To accomplish this, a signal and a slot are required.

```
connect(page()->mainFrame(), SIGNAL(javaScriptWindowObjectCleared()),
           this, SLOT(addBrowserExtension()));
```

The javaScriptWindowObjectCleared() signal is emitted by a QWebFrame automatically when the JavaScript window object is cleared. This in turn calls our own addBrowserExtension() slot.

```
void Browser::addBrowserExtension()
{
    page()->mainFrame()->addToJavaScriptWindowObject(
                         "StorageHandler", m_sh);
}
```

The only thing that this slot does, is re-adding our Qt C++ object back to the loaded webpage. In this manner all pages loaded using this web browser will have access to the functions defined in the 'StorageHandler' class.

The C++ object has to be added as a QML type to a plugin to be available within the QML runtime. In subsection 2.3.2 more information can be found on how to accomplish this.

The other option to have storage access is by using a HTTP REST application service. This is basically a server which can be accessed using the JavaScript XmlHttpRequest API. All storage requests from an application will be handled by this server. Building an application using the XmlHttpRequest API has the advantage that the server can be both on the device or elsewhere.

3.3 Performance comparison

The memory tests used the applications described section 3.1. The source code for these applications can be found in G. All these tests were carried out on both an Nokia N900 and a Linux Maemo SDK virtual machine (Ubuntu 9.04-based) running with 6-cores and 6 GiB of memory enabled on a machine with a hyper-threaded quad-core Intel Core i7-920 CPU and 8 GiB of memory installed. The host-OS was Windows 7 Ultimate x64. Several Qt versions were used for testing, the most recent being commit e43ab8d8 from the 4.7-fremantle branch of the public x11-maemo git repository dated 10 May 2010. There were no significant differences between the Qt versions used (less than 100K difference in memory usage).

The memory performance with QtWebKit was tested using the browser for which the source code can be found in appendices A and B. QML memory performance was tested with the QML runtime executable. Both runtime applications were run using the "-graphicssystem raster" command-line option. Audio did not work on the N900 with the Qt version used, therefore the memory performance results while playing back audio on the N900 were omitted.

Memory usage in kilobytes.

	QtWebKit-PC	QtWebKit-N900	QML-PC	QML-N900
Menu	190864	85076	67988	78604
Contacts	191300	94232	78424	89004
Player - loading	225636		121276	
Player - playing	226056		121692	

Looking at the table above and figure 3.13 a big memory gap between QML and QtWebKit can be seen on the desktop, on the N900 however, this memory gap is much smaller, from more than double on the PC to less than 10% more on the N900. No explanation for this behaviour could be given. Other interesting facts are the increase of memory for QML on the N900 over QML on the desktop and the difference in memory needed to play a media file between QtWebKit and QML. Both technologies use the same media playing backend, but still the memory increase is around 50 megabytes when using QML and around 30 megabytes when using QtWebKit.

The speed performance for QtWebKit was tested in the same browser as the memory performance. For QML a special runtime was developed which implements functionality for speed performance measurement. Source code for this runtime can be found in appendices A and C.

I used the test applications for which the source code can be found in appendix H. These test applications display a rotating green rectangle which should run at a maximum speed of 60 FPS (frames per second). This 60 FPS limit was enforced by forcing the video driver of the test PCś to use vertical synchronization with a display that had a maximum refresh rate of 60 Hz.

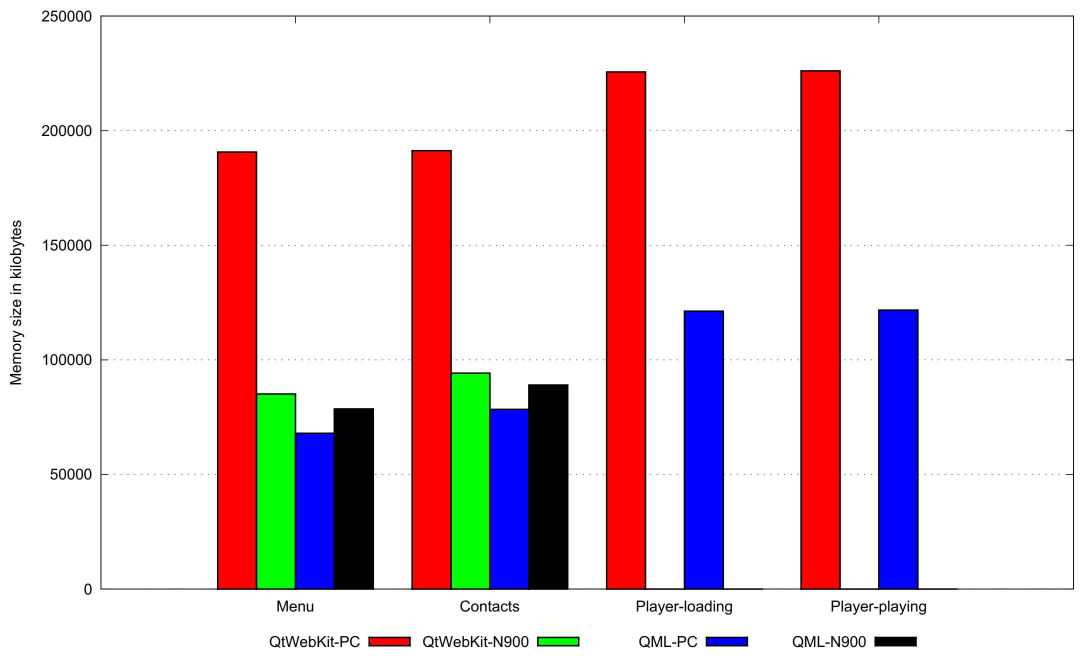

Figure 3.13: Memory performance graph - Graph showing the memory usage of QtWebKit and QML on the desktop and N900.

These tests have however been run on Windows 7 Ultimate (the host OS of the virtual machine), the Linux Virtual Machine, a Nokia N900 and Mac OS X 10.6.3 (MacBook Pro 2.4GHz Core2Duo with 4GiB memory). This has been done to see if there are big performance differences between the different platforms. The "-graphicssystem raster" command-line option has been used in all cases.

Maximum frames per second.

	QtWebKit	QML
Linux	36	60
Mac OS X	53	60
N900	42	60
Windows 7	32	60

Because the maximum FPS was forced to be 60 it is not possible to say if there was a problem with QtWebKit on Windows which caused the QtWebKit application to be so much slower than all other versions, or that Qt itself is slower on Windows than on the other platforms.
QML is faster in all cases, but this might be partly due to the abscence of anti-aliasing. In figure 3.14 it is clearly visible that QtWebKit applies anti-aliasing, while QML does not. There are no options to enable or disable the anti-aliasing in either of the technologies. It is hard to say which technology is the best option due to this aliasing behaviour. In general

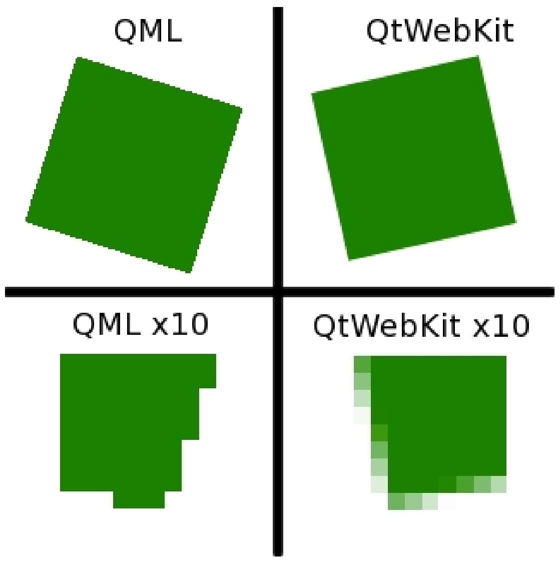

Figure 3.14: Aliasing in QML and QtWebKit - Screenshot of the performance testing applications where the aliasing behaviour of the technologies is clearly visible.

the resolutions on smartphone displays is so high that it is hard to see the aliasing artifacts, which is in favor of QML. In this case, where the contrast between the background and foreground is big, the aliasing artifacts were clearly visible even on the N900 800x480 pixel screen.

The test application used is very simple. It only measures frames per second and not the actual time spent on painting, CPU usage, etc. This application could be extended to draw multiple rectangles to see if the performance drops linearly. The used Qt version can also be extended to be able to profile the actual paint commands, to see how much time is spent on the actual painting, but this is an exercise for the reader.

4

Conclusion

This report has given an overview of the similarities and differences between QML and HTML5 to help to answer the question: *Which of QML/JavaScript or HTML5/JavaScript is a better technology for developing mobile User Interface applications?*. This comparison has been done on several areas, like animation support, code reuseability and memory consumption.

In most areas QtWebKit and QML provide similar functionality. These functionalities are implemented slightly different, but in general they are similar. Functionalities which are different are the animation support, context switching and object placement. QML has a lot more options with regards to animations, including the mixing of sequential and parallel animations and the support for states. These are very important aspects in modern mobile application development which are a big lacking factor when using HTML. The object placement in QML is different in such that the objects are not resized automatically, but this gives the developer more control over how the objects are laid out. In HTML the layout consists mostly of nested boxes which can have undefined sizes if their contents do not fit porperly. Context switching in QML is more elaborate with switching the source on a Loader-element. This does however add the possibility to use animations when switching from one context to the next. This is hard to do using HTML.

Performance-wise QML might be the better choice on processor and memory constrained devices. The interface rendering speed is higher, partly at the cost of aliasing artifacts, and less memory is used by the runtime.

All in all QML is definitely the better choice for mobile use interface development, the performance is better and more options geared towards application development are available. In some use cases where a developer has experience with HTML and the interface is not highly animated, HTML might be a better choice, but QML is the better choice in all other cases.

Glossary

API	Application Programming Interface. An interface defined by a software program to enable the program to interact with other software programs. An API is also defined to specify calling conventions a developer using this program should use to develop his own program,
Application service	An external software application that handles communication with other applications. An example is a webserver.
HTML	Hyper Text Markup Language. The language used to create webpages.
HTML5	Version 5 of HTML which is still in draft as of April 2010. Added functionalities compared to earlier versions include local storage and media playback functionalities.
HTTP REST	HyperText Transfer Protocol Representational State Transfer. A software architecture style which is used for the internet. REST defines several constraints to which a system has to comply to be called RESTful.
JavaScript	A programming language which is used on webpages in combination with HTML and CSS.
JSON	JavaScript Object Notation. A document markup language which can be easily read by JavaScript.
Kano Model	A model developed in 1984 by Professor Noriaki Kano classifying customer preferences
Linux	An Operating System kernel. Most often Linux means the kernel with the GNU tools to form the GNU/Linux Operating System.
Maemo	Nokia's newest Linux-based smartphone operating system.
QML	Qt declarative Markup Language. A JSON like markup language to used to develop Qt declarative interfaces.
qmldir file	A file containing the location of files to be loaded by the QML runtime dynamically.
Qt	A cross-platform application development toolkit developed by Nokia Qt Development Frameworks.
QtWebKit	A part of Qt which adds WebKit with added Qt functionality to the toolkit.
Signal-Slot mechanism	A mechanism within the Qt Toolkit which allows objects to communicate with one another.
Smartphone	A large-screen, data-centric, handheld device designed to offer complete phone functions whilst simultaneously functioning as a personal digital assistant (PDA)
SVG	Scalable Vector Graphics. An open XML-format used for vector graphics.
Toolkit	A collection of tools which simplifies software development.
WebKit	A web rendering engine. Originally a fork of KHTML which was developed by the KDE project.
WRT	Web RunTime. A Nokia framework for developing mobile applications for Nokia devices.
XML	eXtensible Markup Language. A set of rules to digitally encode documents.

References

[1] **Clutter About**. World Wide Web electronic publication. Available from: http://www.clutter-project.org/. 2

[2] **The Home of the Maemo Community**. World Wide Web electronic publication. Available from: http://maemo.org/intro/. 1

[3] **Internet History**. World Wide Web electronic publication. Available from: http://www.computerhistory.org/internet_history/. 9

[4] **Maemo background**. World Wide Web electronic publication. Available from: http://maemo.nokia.com/maemo/. 1

[5] **Maemo features**. World Wide Web electronic publication. Available from: http://maemo.nokia.com/features/. 2

[6] **OpenGL ES Overview**. World Wide Web electronic publication. Available from: http://www.khronos.org/opengles/. 4

[7] **Telepathy the Flexible Communications Framework**. World Wide Web electronic publication. Available from: http://telepathy.freedesktop.org/. 4

[8] **Template:Release history table**. World Wide Web electronic publication. Available from: http://wiki.maemo.org/Template:Release_history_table. 1

[9] **What is GStreamer?** World Wide Web electronic publication. Available from: http://gstreamer.freedesktop.org/. 3

[10] **OS 2008 Features**. World Wide Web electronic publication, 2008. Available from: http://europe.nokia.com/find-products/devices/os2008/features. 2

[11] **Our software strategy**. World Wide Web electronic publication, 2009. Available from: http://phx.corporate-ir.net/External.File?item=UGFyZW50SUQ9MjQxMzB8Q2hpbGRJRD0tMXxUeXBlPTM=&t=1. xi

[12] Tim Berners-Lee. **HTML Tags**. World Wide Web electronic publication, 1991. Available from: http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html. 9

[13] Tim Berners-Lee. **Re: status. Re: X11 BROWSER for WWW**. World Wide Web electronic publication, 1991. Available from: http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html. 9

[14] Tim Berners-Lee. **How It All Started**. World Wide Web electronic publication, 2004. Available from: http://www.w3.org/2004/Talks/w3c10-HowItAllStarted/. 9

[15] Jo Best. **Analysis: What is a smart phone?** World Wide Web electronic publication, 2006. Available from: http://www.silicon.com/technology/mobile/2006/02/13/analysis-what-is-a-smart-phone-39156391/. 1

[16] Jasmin Blanchette and Mark Summerfield. *C++ GUI Programming with Qt 4*, chapter A Brief History of Qt. Prentice Hall, 2006. 7

[17] Bert Bos. **Web Style Sheets**. World Wide Web electronic publication. Available from: http://www.w3.org/Style/. 10

[18] Vannevar Bush. **As We May Think**. *Athlantic Monthly*, **July 1945**, 1945. 9

[19] Stephen Chapman. **A Brief History of Javascript**. World Wide Web electronic publication. Available from: http://javascript.about.com/od/reference/a/history.htm. 13

[20] Linux Devices. **Device Profile: Nokia 770 Internet Tablet**. World Wide Web electronic publication, 2005. Available from: http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Device-Profile-Nokia-770-Internet-Tablet/. 1

[21] Linux Devices. **Nokia 770 Tablet "OS 2006" arrives**. World Wide Web electronic publication, 2006. Available from: http://www.linuxfordevices.com/c/a/News/Nokia-770-Tablet-OS-2006-arrives/. 1

[22] Linux Devices. **New Nokia N800 Internet Tablet available now**. World Wide Web electronic publication, 2007. Available from: http://www.linuxfordevices.com/c/a/News/New-Nokia-N800-Internet-Tablet-available-now/. 1

[23] Douglas Engelbart. **Augmenting Human Intellect: A Conceptual Framework**. World Wide Web electronic publication, 1962. Available from: http://www.invisiblerevolution.net/engelbart/full_62_paper_augm_hum_int.html. 9

[24] Nokia Qt Development Frameworks. **Faster, More Cost-Effective Development Using the Qt Cross-Platform Application & UI Framework**. Whitepaper, 2009. 7

[25] Nokia Qt Development Frameworks. **All Qt Modules**. World Wide Web electronic publication, 2010. Available from: http://doc.trolltech.com/4.6/modules.html. 8

[26] Nokia Qt Development Frameworks. **Build System**. World Wide Web electronic publication, 2010. Available from: http://doc.trolltech.com/4.6/qt4-intro.html#build-system. 8

[27] Nokia Qt Development Frameworks. **Recent Additions to Qt 4**. World Wide Web electronic publication, 2010. Available from: http://doc.trolltech.com/4.6/qt4-intro.html#recent-additions-to-qt-4. 8

[28] Quim Gil. **Maemo Harmattan Qt And More**. World Wide Web electronic publication, 2009. Available from: http://www.slideshare.net/qgil/maemo-harmattan-qt-and-more. 2, 4

[29] Web Design Group. **CSS Structure and Rules**. World Wide Web electronic publication. Available from: http://htmlhelp.com/reference/css/structure.html. 11

[30] Kent Hansen. **Qt State Machine Framework**. World Wide Web electronic publication, 2009. Available from: http://labs.qt.nokia.com/blogs/2009/01/30/qt-state-machine-framework/. 11

[31] Andreas Aardal Hanssen. **Welcome to Kinetic**. World Wide Web electronic publication, 2008. Available from: http://labs.trolltech.com/blogs/2008/11/06/welcome-to-kinetic/. 11

REFERENCES

[32] Andreas Aardal Hanssen. **Animated Tiles**. World Wide Web electronic publication, 2009. Available from: http://labs.trolltech.com/blogs/2009/03/04/animated-tiles/. 11

[33] Ian Hickson. **Web Storage**. World Wide Web electronic publication, 2009. Available from: http://www.w3.org/TR/2009/WD-webstorage-20091222/. 32

[34] Gartner Inc. **Gartner Says Grey-Market Sales and Destocking Drive Worldwide Mobile Phone Sales to 309 Million Units; Smartphone Sales Grew 13 Per Cent in Third Quarter of 2009**. World Wide Web electronic publication, 2009. Available from: http://www.gartner.com/it/page.jsp?id=1224645. xi

[35] Noriaki Kano, Nobuhiku Seraku, Fumio Takahashi, and Shinichi Tsuji. **Attractive quality and must-be quality**. *Journal of the Japanese Society for Quality Control*, **14**:39–48, 1984. 4, 5

[36] Paul Krill. **JavaScript creator ponders past, future**. World Wide Web electronic publication, 2008. Available from: http://www.infoworld.com/d/developer-world/javascript-creator-ponders-past-future-704. 13

[37] Jan-Arve Sæther. **Anchor layout joins the family of QGraphicsLayout**. World Wide Web electronic publication, 2009. Available from: http://labs.trolltech.com/blogs/2009/11/26/anchor-layout-joins-the-family-of-qgraphicslayout/. 11

[38] QtDeclarative Team. **Qt Declarative UI**. World Wide Web electronic publication, 2009. Available from: http://labs.trolltech.com/blogs/2009/05/13/qt-declarative-ui/. 11

[39] The GTK+ Team. **The GTK+ Project Overview**. World Wide Web electronic publication, 2007. Available from: http://www.gtk.org/overview.html. 3

[40] The GStreamer team at http://gstreamer.freedesktop.org/. **gst-launch-0.10(1) - Linux man page**. World Wide Web electronic publication. Available from: http://linux.die.net/man/1/gst-launch-0.10. 3

[41] thoughtfix. **OS2008 "Diablo" update: A pictorial**. World Wide Web electronic publication, 2008. Available from: http://tabletblog.com/2008/06/os2008-diablo-update-pictorial.html. 2

[42] Iago Toral. **MAFW: the Media Application Framework for Maemo**. World Wide Web electronic publication, 2009. Available from: http://www.grancanariadesktopsummit.org/node/219. 4

[43] Judy van Biljon, Paula Kotzé, and Karen Renaud. **Mobile phone usage of young adults: the impact of motivational factors**. In *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*, pages 57–64, 2008. 4, 5

[44] Don Wells. **Extreme Programming: A gentle introduction**. World Wide Web electronic publication, 2009. Available from: http://www.extremeprogramming.org/. xv

[45] Nathan Willis. **Maemo 5, N900 Centerpiece Maemo Summit 2009**. World Wide Web electronic publication, 2009. Available from: http://www.linux.com/news/embedded-mobile/mids/147172-maemo-5-n900-centerpiece-maemo-summit-2009. 2

Appendix A

SubClassedApplication source code

A.1 subclassedapplication.h

```
#ifndef SUBCLASSEDAPPLICATION_H
#define SUBCLASSEDAPPLICATION_H

#include <QApplication>
#include <QTimer>

class SubClassedApplication : public QApplication
{
    Q_OBJECT

    public:
        SubClassedApplication(int argc, char** argv);
        virtual bool notify(QObject *object, QEvent *event);
        void startCounter();

    private:
        int m_frameCounter;
        QTimer *m_timer;

    public slotsL
        void outputFPS();
};

#endif // SUBCLASSEDAPPLICATION_H
```

A.2 subclassedapplication.cpp

```
#include "subclassedapplication.h"

#include <QDebug>

SubClassedApplication::SubClassedApplication(int argc, char **argv)
        : QApplication(argc, argv)
{
        m_frameCounter = 0;
}

void SubClassesApplication::startCointer()
{
    m_timer = new QTimer();
    m_timer->connect(m_timer, SIGNAL(timeout()), this, SLOT(outputFPS()));
    m_timer->start(5000);
}

bool SubClassedApplication::notify(QObject *object, QEvent *event)
{
    if( event->type() == QEvent::Paint )
    {
        m_frameCounter++;
    }
    return QApplication::notify(object, event);
}

void SubClassedApplication::outputFPS()
{
    if( m_frameCounter != 0 )
    {
        qDebug() << m_frameCounter << "frames/5sec:" << m_frameCounter / 5 << "FPS.";
    }
}
```

Appendix B

Browser source code

B.1 main.cpp

```
#include "browser.h"
#include "subclassedapplication.h"

int main(int argc, char *argv[])
{
    SubClassedApplication app(argc, argv);
    new Browser();
    app.startCounter();
    return app.exec();
}
```

B.2 browser.h

```
#ifndef BROWSER_H
#define BROWSER_H

#include <QtGui>
#include <QtWebKit>

#include "filestorageplugin.h"

class Browser : public QWebView
{
    Q_OBJECT

    public:
        Browser();
        ~Browser();

    private:
        FileStoragePlugin *m_fsp;

    private slots:
        void addBrowserExtension();
};

#endif //BROWSER_H
```

B.3 browser.cpp

```
#include "browser.h"

#define APP_WIDTH 800
#define APP_HEIGHT 480

Browser::Browser()
    : m_fsp(new FileStoragePlugin(this))
{
    setFixedSize(APP_WIDTH, APP_HEIGHT);
    setGeometry((qApp->desktop()->width() / 2) - (APP_WIDTH / 2),
                    (qApp->desktop()->height() / 2) - (APP_HEIGHT / 2),
                    APP_WIDTH, APP_HEIGHT);
    QWebSettings::globalSettings()->setAttribute(QWebSettings::PluginsEnabled, 0);

    connect(page()->mainFrame(), SIGNAL(javaScriptWindowObjectCleared()),
            this, SLOT(addFSPlugin()));

    setUrl(QFileDialog::getOpenFileName(this, "Open HTML file",
                                        QDir::homePath(), "HTML Files (*.html)"));

    show();
}

Browser::~Browser()
{
}

void Browser::addBrowserExtension()
{
    page()->mainFrame()->addToJavaScriptWindowObject("FileStoragePlugin", m_fsp);
}
```

Appendix C

Viewer source code

C.1 main.cpp

```
#include "subclassedapplication.h"
#include "viewer.h"

int main(int argc, char *argv[])
{
    SubClassedApplication app(argc, argv);
    new Viewer();
    app.startCounter();
    return app.exec();
}
```

C.2 viewer.h

```
#ifndef VIEWER_H
#define VIEWER_H

#include <QtGui>
#include <QtDeclarative>

class Viewer : public QDeclarativeView
{
    Q_OBJECT

    public:
        Viewer();
        ~Viewer();
};

#endif //Viewer_H
```

C.3 viewer.cpp

```
#include "viewer.h"

#define APP_WIDTH 800
#define APP_HEIGHT 480

Viewer::Viewer()
{
    setFixedSize(APP_WIDTH, APP_HEIGHT);
    setGeometry((qApp->desktop()->width() / 2) - (APP_WIDTH / 2),
                        (qApp->desktop()->height() / 2) - (APP_HEIGHT / 2),
                         APP_WIDTH, APP_HEIGHT);

    QUrl loadedFile = QUrl::fromLocalFile(QFileDialog::getOpenFileName(
                this, "Open QML file", QDir::homePath(), "QML Files (*.qml)"));

    engine()->addImportPath(loadedFile.toString());
    setSource(loadedFile);
    show();
}

Viewer::~Viewer()
{
}
```

Appendix D

Storage handler source code

D.1 storagehandler.h

```
#ifndef STORAGEHANDLER_H
#define STORAGEHANDLER_H

#include <QtSql>

class StorageHandler : public QObject
{
    Q_OBJECT

    public:
        StorageHandler(QObject* parent = 0);
        ~StorageHandler();

    private:
        QSqlDatabase m_db;
        QSqlQuery m_query;

    public slots:
        int initializeDatabase(QVariant data);
        QString loadFromDatabase();
        void saveToDatabase(QVariant query, QVariantList values);
};

#endif //STORAGEHANDLER_H
```

D.2 storagehandler.cpp

```
#include "storagehandler.h"

StorageHandler::StorageHandler(QObject* parent)
{
}

StorageHandler::~StorageHandler()
{
    m_db.close();
}

int StorageHandler::initializeDatabase(QVariant data)
{
    m_db = QSqlDatabase::addDatabase("QSQLITE");
    m_db.setDatabaseName(QDir::tempPath() + "/" + data.toString());
    bool ok = m_db.open();
    if(ok)
    {
        m_query = QSqlQuery(m_db);
    }
    return ok;
}

QString StorageHandler::loadFromDatabase()
{
    m_query.exec("SELECT * FROM contacts");
    int nameNo = m_query.record().indexOf("name");
    int phoneNo = m_query.record().indexOf("phonenumber");
    QString returnString;
    while (m_query.next()) {
        returnString.append(m_query.value(nameNo).toString() + ": " + \
        m_query.value(phoneNo).toString() + \
        "\n<br />\n");
        /* "\n<br />\n" is used for HTML to insert a line-break, when using
         * QML only "\n" is needed. This is the only difference in code
         * between the QtWebKit and QML versions of this handler.
         */
    }
    return returnString;
}

void StorageHandler::saveToDatabase(QVariant query, QVariantList values)
{
    m_query.prepare(query.toString());
    int counter = 0;
    while (counter < values.length())
    {
        m_query.addBindValue(values.at(counter));
        counter++;
    }
    m_query.exec();
}
```

Appendix E

<div> table source code

```
<html>
<head>
</head>
<body>
<style type="text/css">
    div
    {
        border-style: solid;
        border-width: 1px;
    }
    .tablewrapper
    {
        border: none;
        position: relative;
    }
    .table
    {
        display: table;
    }
    .row
    {
        border: none;
        display: table-row;
    }
    .cell
    {
        width: 150px;
        display: table-cell;
    }
    .cell.empty
    {
        border: none;
        width: 150px;
    }
    .cell.colspanned
    {
        width: 300px;
    }
    .cell.subcelled
    {
        width: 150px;
    }
    .cell.rowspanned
    {
        position: absolute;
        top: 1;
        bottom: 1;
        width: 150px;
    }
```

E. <DIV> TABLE SOURCE CODE

```
</style>
<div class="tablewrapper">
    <div class="table">
        <div class="row">
            <div class="cell">&nbsp;</div>
            <div class="colspanned cell">&nbsp;</div>
            <div class="rowspanned cell">&nbsp;</div>
        </div>
        <div class="row">
            <div class="cell">&nbsp;</div>
            <div class="cell" style="border:none;">
                <div class="subcelled cell">&nbsp;</div>
                <div class="subcelled cell">&nbsp;</div>
            </div>
            <div class="empty cell"></div>
        </div>
    </div>
</div>
</body>
</html>
```

Appendix F

QML table layout

```
import Qt 4.7

Item
{
    width: 602
    height: 202

    Rectangle
    {
        border.width: 2
        height: 100
        id: topLeft
        width: 100
        x: 100
        y: 1
    }

    Rectangle
    {
        anchors.left: topLeft.right
        anchors.top: topLeft.top
        border.width: 2
        height: topLeft.height
        id: topCenter
        width: topLeft.width * 2
    }

    Rectangle
    {
        anchors.left: topCenter.right
        anchors.top: topLeft.top
        border.width: 2
        height: topLeft.height * 2
        id: topRight
        width: topLeft.width
    }

    Rectangle
    {
        anchors.left: topLeft.left
        anchors.top: topLeft.bottom
        border.width: 2
        height: topLeft.height
        id: bottomLeft
        width: topLeft.width
    }

    Rectangle
    {
        anchors.left: bottomLeft.right
```

```
            anchors.top: topLeft.bottom
            border.width: 2
            height: topLeft.height
            id: bottomLeftCenter
            width: topLeft.width
        }

        Rectangle
        {
            anchors.left: bottomLeftCenter.right
            anchors.top: topLeft.bottom
            border.width: 2
            height: topLeft.height
            id: bottomRightCenter
            width: topLeft.width
        }

    }
```

Appendix G

Memory tests source code

G.1 Menu (V3)

G.1.1 HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Main menu</title>
<style type="text/css">
.wraptocenter {
    display: table-cell;
    text-align: center;
    vertical-align: middle;
    width: 800px;
    height: 430px;
}
.wraptocenter * {
    vertical-align: middle;
}
</style>
</head>
<body>
    <div class="wraptocenter">
        <center>
        <table>
            <tr>
                <td align="center">
                    <a href="../Contacts/contacts.html"><img src="images/contacts.png" /></a>
                    <br />
                    <h2>
                        Contacts
                    </h2>
                </td>
                <td  align="center">
                    <a href="../Player/player.html"><img src="images/player.png" /></a>
                    <br />
                    <h2>
                        Player
                    </h2>
                </td>
            </tr>
        </table>
        </center>
    </div>
</body>
</html>
```

G.1.2 QML

G.1.2.1 Menu.qml

```
import Qt 4.7

Loader
{
    property url nextSource

    id: rootLoader

    width: 800
    height: 480

    resizeMode: Loader.SizeItemToLoader
    source: "Contents.qml"

    SequentialAnimation
    {
        id: animation
        NumberAnimation { target: rootLoader; property: "opacity"; to: 0; }
        ScriptAction { script: source = nextSource; }
        NumberAnimation { target: rootLoader; property: "opacity"; to: 1; }
    }
}
```

G.1.2.2 Contents.qml

```
import Qt 4.7

Item
{
    Row
    {
        anchors.horizontalCenter: parent.horizontalCenter;
        anchors.verticalCenter: parent.verticalCenter

        MenuItem
        {
            id: firstItem
            image: "images/contacts.png"
            fileSource: "../Contacts/Contacts.qml"
            text: "Contacts"
        }

        MenuItem
        {
            anchors.left: firstItem.right
            image: "images/player.png"
            fileSource: "../Player/Player.qml"
            text: "Player"
        }

    }
}
```

G.1.2.3 MenuItem.qml

```
import Qt 4.7

Item
{
    id: itemRoot

    property alias image: itemImage.source
    property alias text: itemText.text

    property string fileSource
```

```
        width: itemImage.width
        height: itemImage.height + itemText.height

        Image
        {
            id: itemImage

            MouseArea
            {
                acceptedButtons: Qt.LeftButton | Qt.RightButton
                anchors.fill: parent
                hoverEnabled: true
                onClicked:
                {
                    rootLoader.nextSource = itemRoot.fileSource;
                    animation.start();
                }
            }
        }

        Text
        {
            anchors.horizontalCenter: itemImage.horizontalCenter
            anchors.top: itemImage.bottom
            font.bold: true; font.pixelSize: 24
            id: itemText
            text: "Contacts"
        }
    }
```

G.2 Contacts (V3)

G.2.1 HTML

G.2.1.1 contacts.html

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="contactsStyle.css" />
    <script type="text/javascript" src="contactsCode.js" />
</head>
<body>
    <div id="topBar" class="topandbottom">
        <div class="close">
            <a href="javascript:javascript:history.go(-1)">
                <img src="images/close.png" border="0px" />
            </a>
        </div>
</div>
<div id="dbContents">Shown when not using special QtWebKit browser.</div>
<div id="bottomBar" class="topandbottom">
    <div class="button">
        <div class="buttonContent" onclick="startContactAdder();">
            Add Contact
        </div>
    </div>
</div>
</body>
</html>
```

G.2.1.2 contactsCode.js

```
function addContact()
{
    var nameInput = document.getElementById('nameInput');
```

```
        var phoneInput = document.getElementById('phoneInput');
        if(nameInput.value == null || nameInput.value == "" ||
                phoneInput.value == null || phoneInput.value == "")
        {
            alert("Please fill in all fields");
        }
        else
        {
            var query = "INSERT into contacts(name, phonenumber) values (?, ?)";
            var values = [nameInput.value, phoneInput.value];
            FileStoragePlugin.saveToDatabase(query, values);
        }
        document.body.removeChild(document.getElementById('contactAdder'));
        fillPage();
    }

    function cancelAdd()
    {
        document.body.removeChild(document.getElementById('contactAdder'));
        fillPage();
    }

    function fillPage()
    {
        document.getElementById("dbContents").innerHTML = FileStoragePlugin.loadFromDatabase();
    }

    function startContactAdder()
    {
        var contactAdder = document.createElement('div');
        contactAdder.setAttribute("id", "contactAdder");
        document.body.appendChild(contactAdder);
        document.getElementById('contactAdder').innerHTML= "<table><tr><td> \
            Name:</td><td><input id=\"nameInput\" type=text size=20  maxlength='128' />\
            </td></tr><tr><td>Phone number:</td><td>\
            <input id=\"phoneInput\" type=text size=20 maxlength=\"20\"/>\
            </td></tr></table><div id=\"bottomBar\" class=\"topandbottom\">\
            <div class=\"button\">\
            <div class=\"buttonContent\" onclick=\"addContact();\">\
            Accept</div></div><div class=\"button\">\
            <div class=\"buttonContent\" onclick=\"cancelAdd();\">\
            Cancel</div></div></div>\n";
    }

    function startDB()
    {
        var succeeded = FileStoragePlugin.initializeDatabase('Contacts');
        if(succeeded == 0)
        {
            alert("Database initialization failed.");
        }
        fillPage();
    }

    window.onload = function()
    {
        startDB();
    }
```

G.2.1.3 contactsStyle.css

```
    body { font-family: Arial, Helvetica, sans-serif }

    td
    {
        padding:0px;
        vertical-align:top;
    }

    .button
```

```
{
    border:3px solid black;
    float:right;
    height:43px;
    margin-bottom:5px;
    margin-right:5px;
    position:relative;
    width:225px;
}

.buttonContent
{
    font-weight:bold;
    font-size:32px;
    height:43px;
    margin-top:-20px;
    position:absolute;
    text-align:center;
    top:50%;
    width:100%;
}

.close
{
    margin-right:5px;
    margin-top:5px;
}

.topandbottom
{
    background-color:#FFF;
    height:53px;
    position:fixed;
    text-align:right;
    width: 100%;
}

#bottomBar
{
    bottom:0px;
    position:fixed;
    right:0px;
}

#contactAdder
{
    bottom:0;
    background-color:#FFF;
    font-size:28px;
    height:6em;
    left:0;
    position: fixed;
    width:100%;
}

#dbContents
{
    font-size:36px;
    font-weight:bold;
    margin:0px;
    padding:0px;
    top:0px;
}

#topBar
{
    top:0px;
    position:fixed;
    left:0px;
}
```

G.2.2 QML

G.2.2.1 Contacts.qml

```
import Qt 4.7
import "importCore" 1.0

Item
{
    FileStoragePlugin
    {
        id: fileStoragePlugin;
        Component.onCompleted: {
            initializeDatabase("Contacts");
            contactsText.text = fileStoragePlugin.loadFromDatabase();
        }
    }

    Image
    {
        anchors.right: parent.right
        anchors.rightMargin: 5
        anchors.top: parent.top
        anchors.topMargin: 5
        id: closeImage
        source: "images/close.png"

        MouseArea
        {
            anchors.fill: parent
            onClicked:
            {
                rootLoader.nextSource = "../Menu/Contents.qml"
                animation.start();
            }
        }
    }

    Button
    {
        anchors.bottom: parent.bottom
        anchors.right: parent.right
        id: addButton
        onClicked:
        {
            adder.opacity = 100;
        }
        text: "Add Contact"
    }

    Flickable
    {
        anchors.bottom: addButton.top
        anchors.top: closeImage.bottom
        clip: true
        contentHeight: contactsText.height
        height: parent.height - closeImage.height
        id: mainFlickable
        width: parent.width

        Text
        {
            font.bold: true
            font.pixelSize: 32
            id: contactsText
            text: ""
        }
    }

    ContactAdder
    {
        anchors.bottom: parent.bottom
        height: 160
        width: parent.width
        id: adder
        opacity: 0
    }
}
```

G.2.2.2 ContactAdder.qml

```
import Qt 4.7

Rectangle
{
    color: "white"

    Text
    {
        anchors.leftMargin: 25
        font.pixelSize: 32
        id: nameText
        text: "Name:"
    }

    Text
    {
        anchors.leftMargin: 25
        anchors.top: nameText.bottom
        font.pixelSize: 32
        id: numberText
        text: "Phone number:"
    }

    Rectangle
    {
        anchors.horizontalCenter: phoneField.horizontalCenter
        anchors.verticalCenter: nameText.verticalCenter
        border.width: 3
        height: 43
        id: nameField
        width: 250

        TextEdit
        {
            anchors.fill: parent
            anchors.leftMargin: 5
            font.pixelSize: 32
            id: nameEdit
        }
    }

    Rectangle
    {
        anchors.verticalCenter: numberText.verticalCenter
        border.width: 3
        height: 43
        id: phoneField
        width: 250
        x: 250

        TextEdit
        {
            anchors.fill: parent
            anchors.leftMargin: 5
            font.pixelSize: 32
            id: phoneEdit
        }
    }

    Button
    {
        anchors.bottom: parent.bottom
        anchors.right: parent.right
        id: acceptButton
```

```
            onClicked:
            {
                var query = "INSERT into contacts(name, phonenumber) values (?, ?)";
                var values = [nameEdit.text, phoneEdit.text];
                fileStoragePlugin.saveToDatabase(query, values);
                parent.opacity = 0;
                nameEdit.text = "";
                phoneEdit.text = "";
                contactsText.text = fileStoragePlugin.loadFromDatabase();
            }
            text: "Accept"
        }

        Button
        {
            anchors.bottom: parent.bottom
            anchors.right: acceptButton.left
            onClicked:
            {
                parent.opacity = 0;
                nameEdit.text = "";
                phoneEdit.text = "";
            }
            text: "Cancel"
        }
    }
```

G.2.2.3 Button.qml

```
import Qt 4.7

Rectangle
{
    property alias text: innerText.text
    signal clicked

    anchors.bottomMargin: 5
    anchors.rightMargin: 5
    border.color: "black"
    border.width: 3
    height: 43
    id: container
    width: 225

    MouseArea
    {
        anchors.fill: parent
        onClicked: container.clicked()
    }

    Text
    {
        anchors.centerIn: parent
        font.family: "Arial"
        font.pixelSize: 32
        id: innerText
    }
}
```

G.3 Player (V3)

G.3.1 HTML

G.3.1.1 player.html

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Main menu</title>
    <link rel="stylesheet" type="text/css" href="playerStyle.css" />
    <script type="text/javascript" src="playerCode.js"></script>
</head>
<body>
    <div id="topBar" class="topBar-landscape">
        <div class="close">
            <a href="javascript:javascript:history.go(-1)">
                <img src="images/close.png" border="0px" />
            </a>
        </div>
    </div>
    <div id="content">
        <div class="wrapmiddletocenter">
            Audio filename
            <div id="timer">
                00:00
            </div>
        </div>
        <div class="wrapbottomtocenter" id="bottomBar">
            <div class="buttonContainer">
                <img src="images/back.png" id="backIcon"
                    onmousedown="startTimer('back');"
                    onmouseup="stopTimer('back');"
                    width="100%" height="100%" />
            </div>
            <div class="buttonContainer">
                <img src="images/play.png" id="playerIcon"
                    onclick="playpause();" width="100%"
                    height="100%" />
            </div>
            <div class="buttonContainer">
                <img src="images/forward.png" id="forwardIcon"
                    onmousedown="startTimer('forward');"
                    onmouseup="stopTimer('forward');"
                    width="100%" height="100%" />
            </div>
         </div>
    </div>
<audio src="File.wav" id="audioElement" ontimeupdate="time()"></audio>
</body>
</html>
```

G.3.1.2 playerCode.js

```
var playing = 0;

function playpause()
{
    var image = document.getElementById("playerIcon");
    var audio = document.getElementById("audioElement");

    if (!playing)
    {
        image.src = "images/pause.png";
        audio.play();
    }
    else
    {
        audio.pause();
        image.src = "images/play.png";
    }
    playing = !playing;
}

function back()
{
    var audio = document.getElementById("audioElement");
    if(audio.currentTime > 1)
    {
        audio.currentTime = audio.currentTime - 1;
    }
}

function forward()
{
    var audio = document.getElementById("audioElement");

    if(audio.currentTime < audio.duration - 1)
    {
        audio.currentTime = audio.currentTime + 1;
    }
}

function gotolandscape()
{
    var topbar = document.getElementById("topBar");
    topbar.className = "topBar-landscape";
}

function gotoportrait()
{
    var topbar = document.getElementById("topBar");
    topbar.className = "topBar-portrait";
}

function startTimer(text)
{
    if(text == "forward")
    {
        forwardTimer = setInterval("forward()", 250);
    }
    else if(text == "back")
    {
        backTimer = setInterval("back()", 250);
    }
}

function stopTimer(text)
{
    if(text == "forward")
    {
        clearInterval(forwardTimer);
    }
    else if(text == "back")
    {
        clearInterval(backTimer);
    }
}

function time()
{
    var audio = document.getElementById("audioElement");
    var timer = document.getElementById("timer");

    var sec = Math.round(audio.currentTime);
    var min = Math.floor(sec/60);
    sec = sec % 60;
    t = two(min) + ":" + two(sec);

    timer.innerHTML = t;
}

function two(x) {return ((x>9)?"":"0")+x}
```

G.3.1.3 playerStyle.css

```
body { font-family: Arial, Helvetica, sans-serif }

div {
    -webkit-transition-property: opacity;
    -webkit-transition-duration: 2s;
}

.buttonContainer
{
    float: left;
    width: 192px;
    height: 192px;
}

.close
{
    margin-left: 5px;
    margin-right: 5px;
    margin-top: 5px;
}

.topBar-landscape
{
    text-align:right;
}

.topBar-portrait
{
    text-align:left;
}

.wrapbottomtocenter
{
    display:table-cell;
    text-align:center;
    vertical-align:middle;
    width: 692px;
    height: 200px;
}

.wrapmiddletocenter
{
    display:table-cell;
    font-size:32px;
    text-align:center;
    width:800px;
    height:200px;
}

#content
{
    bottom:0px;
    position:fixed;
    right:0px;
    height: 75%;
}

#bottomBar
{
    bottom:0px;
    position:fixed;
    right:0px;
    height:192px;
}

#timer
{
    display:table-cell;
    text-align:center;
    vertical-align:bottom;
    width:800px;
    height:150px;
}

#topBar
{
    top:0px;
    position:fixed;
    left:0px;
    background-color:#FFF;
    height:53px;
    position:fixed;
    width: 100%;
}
```

G.3.2 QML

G.3.2.1 Player.qml

```
    import Qt 4.7
    import Qt.multimedia 4.7
    import "playerCode.js" as Code

    Item
    {
        height: 480
        id: main
        width: 800

        Image
        {
            anchors.leftMargin: 5
            anchors.right: parent.right
            anchors.rightMargin: 5
            anchors.top: parent.top
            anchors.topMargin: 5
            id: closeImage
            source: "images/close.png"

            MouseArea
            {
                anchors.fill: parent
                onClicked:
                {
                    rootLoader.nextSource = "../../Menu/V3/Contents_2.qml"
                    animation.start();
                }
            }
        }

        Item
        {
            anchors.centerIn: parent
            height: 480
            id: container
            state: (runtime.orientation == Orientation.Portrait) ? '' : 'Portrait'
            width: 800

            Item
            {
                anchors.bottom: parent.bottom
                height: parent.height * 0.75
                id: playerControls
                width: parent.width

                Text
                {
                    anchors.horizontalCenter: parent.horizontalCenter
                    font.pixelSize: 32
                    id: urlText
                    text: "Audio filename"
                }

                Text
                {
                    anchors.horizontalCenter: parent.horizontalCenter
                    anchors.bottom: playerImage.top
                    font.pixelSize: 32
                    id: timeText
                    text: Code.time(audioElement.position);
                }

                Image
                {
                    anchors.right: playerImage.left
                    anchors.verticalCenter: playerImage.verticalCenter
                    height: playerImage.height
                    id: backImage
                    source: "images/back.png"
                    width: playerImage.width

                    MouseArea
                    {
                        anchors.fill: parent
                        onPressed:
                        {
                            Code.startTimer("back");
                        }
                        onReleased:
                        {
                            Code.stopTimer("back");
                        }
                    }
                }

                Image
                {
                    anchors.bottom: parent.bottom
                    anchors.horizontalCenter: parent.horizontalCenter
                    id: playerImage
                    source: "images/play.png"

                    MouseArea
                    {
                        anchors.fill: parent
                        onClicked:
                        {
                            Code.playpause();
                        }
                    }
                }

                Image
                {
                    anchors.left: playerImage.right
                    anchors.verticalCenter: playerImage.verticalCenter
                    height: playerImage.height
                    id: forwardImage
                    source: "images/forward.png"
                    width: playerImage.width

                    MouseArea
                    {
                        anchors.fill: parent
                        onPressed:
                        {
                            Code.startTimer("forward");
                        }
                        onReleased:
                        {
                            Code.stopTimer("forward");
                        }
                    }
                }
            }

            Audio
            {
                id: audioElement
                source: "../Information_First.wav"
            }
```

```
            states:
                State
                {
                    name: "Portrait"


                    PropertyChanges
                    {
                        target: container
                        rotation: -90
                        width: 480
                        height: 800
                    }
                    PropertyChanges
                    {
                        target: playerImage
                        width: 128
                        height: 128
                    }
                    AnchorChanges
                    {
                        target: closeImage
                        anchors.right: undefined
                        anchors.left: main.left
                    }
                }
            transitions:
                Transition
                {
                    from: ""
                    to: "Portrait"
                    reversible: true

                    ParallelAnimation
                    {
                        AnchorAnimation {}
                        NumberAnimation
                        {
                            properties: "rotation, width, height, x"
                            duration: 500
                            easing.type: "InOutBack"
                        }
                    }
                }
        }
    }
```

G.3.2.2 playerCode.js

```
var playing = 0;
var t = 0;
var forwardTimer;
var backTimer;

function playpause()
{
    if (!playing)
    {
        playerImage.source = "images/pause.png";
        audioElement.play();
    }
    else
    {
        audioElement.pause();
        playerImage.source = "images/play.png";
    }
    playing = !playing;
}

function back()
{
    if(audioElement.position > 1000000)
    {
        audioElement.position = audioElement.position - 1000000;
    }
}

function forward()
{
    if(audioElement.position < audioElement.duration - 1000000)
    {
        audioElement.position = audioElement.position + 1000000;
    }
}

function startTimer(text)
{
    if(text == "forward")
    {
        forwardTimer = setInterval("forward()", 250);
    }
    else if(text == "back")
    {
        backTimer = setInterval("back()", 250);
    }
}

function stopTimer(text)
{
    if(text == "forward")
    {
        clearInterval(forwardTimer);
    }
    else if(text == "back")
    {
        clearInterval(backTimer);
    }
}

function two(x) {return ((x>9)?"":"0")+x}

function time(ms)
{
    var sec = Math.floor(ms/1000)
    ms = ms % 1000

    var min = Math.floor(sec/60)
    sec = sec % 60
    t = two(sec)

    min = min % 60
    t = two(min) + ":" + t

return t
}
```

Appendix H

Performance test source code

H.1 HTML

```
<!DOCTYPE html>
<html>
    <head>
        <style type="text/css">
            #rotatediv
            {
                -webkit-animation-name: rotateThis;
                -webkit-animation-duration: 2s;
                -webkit-animation-iteration-count: infinite;
                -webkit-animation-timing-function: linear;
                background-color: green;
                height: 100px;
                left: 100px;
                position: absolute;
                top: 100px;
                width: 100px;
            }

            @-webkit-keyframes rotateThis
            {
                from
                {
                    -webkit-transform:rotate(0deg);
                }
                to
                {
                    -webkit-transform:rotate(360deg);
                }
            }
        </style>
    </head>
    <body>
        <div id="rotatediv">&nbsp;</div>
    </body>
</html>
```

H.2 QML

```
import Qt 4.7

Item
{
    height: 800
    width: 480

    Rectangle
    {
        color: "green"
        height: 100
        width: 100
        x: 100
        y: 100

        RotationAnimation on rotation
        {
            duration: 2000
            loops: Animation.Infinite
            from: 0
            to: 360
        }
    }
}
```