

# **Comparing Google's Android and Apple's iOS Mobile Software Development Environments**

Eero Maasalmi, Panu Pitkänen

Thesis

Business Information Technology

2011



Business Information Technology

<p><b>Author or authors</b> Eero Maasalmi, Panu Pitkänen</p>	<p><b>Group or year of entry</b> 2008</p>
<p><b>Title of report</b> Comparing Google's Android and Apple's iOS Mobile Software Development Environments</p>	<p><b>Number of pages and appendices</b> 104 + 6</p>
<p><b>Supervisors</b> Ismo Harjunmaa, Juhani Välimäki</p>	
<p>Mobile devices have become extremely popular during the past few years. They are used widely in business and everyday life by the young and the elderly. As the mobile devices and their operating systems have developed, the manufacturers have made it possible also for everyday users to create their own applications using specific Software Development Kits. For that reason, it is now common that applications are created not only by third party companies but also by everyone interested in the matter.</p> <p>The mobile business has come to a point where there are a few big companies responsible for developing the operating systems used by most hardware manufacturers. Of all the operating systems, there are two which have grown their market share during the past few years: Apple's iOS and Google's Android. The purpose of this thesis was to compare the two, finding out how easy they are to take into use, and to develop and publish applications with.</p> <p>The study was carried out as an empirical research. The research was made on both operating systems and the SDKs. Based on that knowledge, applications were created and published for both systems. The basic outline of the study was installing and working with both SDKs, developing and publishing applications using the SDKs, and estimating the costs of installing development kits in an educational environment.</p> <p>The objectives of this study were achieved as planned: both SDKs were successfully installed, four applications were created altogether, an estimation of costs was made and overall experience of both systems was gained.</p>	
<p><b>Keywords</b> Android, Apple iOS, Software Development Kit, mobile device</p>	

## Table of contents

1	Introduction.....	1
1.1	Objective of the thesis.....	1
1.2	Outline of the thesis.....	2
1.3	Scope of the thesis .....	2
1.4	Out of scope .....	2
1.5	Methodology applied.....	3
2	Android.....	4
2.1	Android and the Open Handset Alliance .....	4
2.2	Android from a technical perspective .....	5
2.2.1	Applications.....	5
2.2.2	Application Framework.....	5
2.2.3	Libraries .....	6
2.2.4	Android Runtime.....	6
2.2.5	Linux Kernel .....	7
2.3	Introduction to Android SDK .....	7
2.4	System and software requirements for developing with Android SDK.....	8
2.4.1	Supported Operating Systems .....	8
2.4.2	Supported Development Environments .....	8
2.4.3	Hardware requirements .....	9
2.5	Installing the Android SDK.....	9
2.6	Summary of the Android SDK installation process.....	26
2.7	Programming with the Android SDK.....	27
2.7.1	Setting up the Android Virtual Device.....	27
2.7.2	Starting the project in the Eclipse environment.....	29
2.7.3	Developing HelloWorld for Android .....	31
2.7.4	Developing OpenSomeWebsite for Android.....	32
2.7.5	Summary of the application development process on Android SDK ...	33
2.8	Deploying an application on Android.....	33
2.8.1	Testing the application on Android SDK.....	34
2.8.2	Considering the use of license agreements .....	35

2.8.3	Giving the application an icon, a label and a correct version number ...	35
2.8.4	Turning off unnecessary facilities and removing unnecessary files .....	37
2.8.5	Generating a cryptographic key and considering the use of Maps API Key .....	37
2.8.6	Compiling, signing and aligning the application.....	38
2.8.7	Releasing the application into the Android Market.....	40
2.9	Legal issues and costs of setting up an Android SDK environment .....	41
3	iOS.....	43
3.1	iOS in general.....	43
3.2	iOS from a technical perspective .....	44
3.2.1	Core OS and Core Services layers.....	45
3.2.2	Media layer.....	45
3.2.3	Cocoa Touch layer.....	46
3.3	iOS Developer Programs .....	46
3.4	Introduction to iOS SDK .....	47
3.5	System and software requirements for developing with iOS SDK.....	48
3.6	Installing the iOS SDK.....	49
3.7	Summary of the iOS SDK installation process.....	56
3.8	Programming with the iOS SDK.....	57
3.8.1	Setting up the virtual iPhone device .....	58
3.8.2	Starting the project in the Xcode environment.....	59
3.8.3	Developing HelloWorld for iOS .....	61
3.8.4	Developing OpenSomeWebsite for iOS .....	62
3.8.5	Summary of the application development process on iOS SDK .....	63
3.9	Deploying an application on iPhone .....	63
3.9.1	Designating iOS devices for development and user testing .....	65
3.9.2	Creating a Certificate Signing Request and obtaining the iOS Development Certificate .....	66
3.9.3	Creating and downloading a development provisioning profile.....	73
3.9.4	Considering the use of license agreements .....	73
3.9.5	Giving the application an icon, a label and a correct version number ...	74
3.9.6	Archiving and signing the application .....	76

3.9.7	Releasing the application into the App Store .....	76
3.10	Legal issues and costs of setting up an iOS SDK environment .....	77
3.10.1	Legal issues .....	77
3.10.2	Costs of setting up an iOS SDK environment.....	78
4	Overall comparison based on the research.....	81
4.1	Pros and cons of using Android as a platform.....	81
4.2	Pros and cons of using iOS as a platform.....	83
5	Discussion about the research.....	86
5.1	Evaluating the methods.....	86
5.2	Evaluating the research results and their validity.....	86
5.3	Evaluating the overall thesis process .....	87
5.4	Evaluating the learning process.....	88
6	Summary and conclusion of the thesis.....	89
6.1	Summary.....	89
6.2	Conclusion .....	89
Bibliography.....		91
General sources used.....		91
Sources used in the research about Android and Android SDK .....		91
Sources used in the research about Apple iOS and iOS SDK.....		94
Appendix A - Android source code .....		98
Appendix A.1. - HelloWorld for Android .....		98
Appendix A.2. - OpenSomeWebsite for Android.....		99
Appendix B - iOS source code.....		100
Appendix B.1. - HelloWorld for iOS .....		100
Appendix B.2. - OpenSomeWebsite for iOS .....		102

## Abbreviations

2D	Two dimensional
3D	Three dimensional
ADB	Android Debug Bridge
ADT	Android Development Tools
API	Application Programming Interface
AVD	Android Virtual Device
CA	Certificate Authority
CRL	Certificate Revocation List
CSS	Cascading Style Sheets
CSR	Certificate Signing Request
DDMS	Dalvik Debug Monitor Server
DPI	Dots Per Inch
DVI	Digital Visual Interface
EFF	Electronic Frontier Finland
EULA	End User License Agreement
GB	Gigabyte
GCC	GNU Compiler Collection
GHz	Gigahertz
GNU GPL	GNU General Public License
GUI	Graphical User Interface
HD	Hard Drive/High Definition
HDMI	High Definition Multimedia Interface
HTML	Hypertext Markup Language
ID	Identification
IDE	Integrated Development Environment
iOS	Apple's mobile operating system (known as iPhone OS before June 2010)
JDK	Java Development Kit
JDT	Java Development Tools
JRE	Java Runtime Environment
MB	Megabyte
OCSP	Online Certificate Status Protocol
OHA	Open Handset Alliance
OS	Operating System

RSA	Rivest, Shamir and Adleman, the designers of the RSA signing/encryption algorithm
SDK	Software Development Kit
SQL	Structured Query Language
UDID	Unique Device Identifier
UI	User Interface
USB	Universal Serial Bus
WWDR	Apple Worldwide Developer Relations Intermediate Certificate
XML	eXtensible Markup Language

# 1 Introduction

Mobile devices are not in any way as they used to be in the past. Not only have the screens grown in size and quality, but also the internal hardware has grown to reach performance levels seen only in laptop computers some years ago. In addition to traditional mobile phones, the market has seen the rise of devices with screen up to over 10 inches, so called tablets. All of this opens doors for new, bigger, faster, better looking and possibly yet never seen applications to be developed.

Google Android and Apple iOS being among the biggest players in the mobile operating system market, there is also a need for usable environments in which more or less experienced developers can create applications of their own for these specific environments. As the mobile application development is luring more and more developers into the market it has also become an attractive topic in educational environment. The growing popularity of Android and iOS has made them the two most interesting platforms for now.

## 1.1 Objective of the thesis

The objective of this thesis is to compare the Android Software Development Kit and Apple's iOS Software Development Kit with each other. We will install both development kits on workstations, create applications with them, research the publishing process of the application and finally find out how much each environment would cost in educational use. All of these steps are documented and the documentation is presented in this thesis.

As this thesis work is done as a pair work, the workload had to be divided evenly before starting in order for both of us to be able to receive our own grade for the work. We divided it so that Eero Maasalmi is responsible for everything related to Apple iOS SDK and its documentation and likewise Panu Pitkänen is responsible for Android SDK and the documentation related to it. We are working as a team, but still keeping our own areas in mind throughout the process.



## 1.2 Outline of the thesis

The basic outline of this thesis work is as follows:

- Introduction.
- Research related to the Android SDK and Android programming.
- Research related to the Apple iOS SDK and iOS programming.
- Conclusion and appendices.

## 1.3 Scope of the thesis

The scope of this thesis work can be listed as follows:

- Installing and familiarizing ourselves with Android SDK and iOS SDK.
- Creating a small application with Android SDK and iOS SDK according to requirements commissioned to us.
- Publishing the application in the operating system's application market.
- Researching the estimated cost of installing either development kit in an educational environment.

## 1.4 Out of scope

This thesis work will not introduce nor explain the very basics of Android and Apple iOS. The reader is expected to have basic knowledge about both operating systems.

We will not be considering the advantages or disadvantages of using the operating systems and applications on tablet-devices and their bigger screens. We are only working with two mobile devices: iPhone 4 and Samsung Galaxy S.

Due to monetary issues, we will not actually be able to publish our application neither in neither Apple's App Store nor in the Android Market. We will describe the publishing process in theory instead.

We will not take sides on which operating system or development environment is better, we will only document the usability and differences.

We will also only consider the creation and publishing of an application; we will not take into consideration whether the application should cost any amount of money and whether the development is done as a profession or merely for own advance.

## **1.5 Methodology applied**

This thesis is carried out as a research project and is partly based on theoretical and partly on empirical research; the systems are first taken into use, and the results will then be documented. It is a research project comparing two development environments intended for similar use. Other source material will be gathered from related literature and from the companies responsible for the systems.

The thesis includes some application development work, which falls under product oriented research. Nevertheless, it is only a minor part of the entity and thus makes this a research-oriented thesis following the contents and methods of the related guideline “Thesis report guidelines for a research project”. For the layout standard, the HAAGA-HELIA UAS Report Layout and Citing Sources will be used as the guideline. (HAAGA-HELIA University of Applied Sciences 2011a, HAAGA-HELIA University of Applied Sciences 2011b.)

## 2 Android

Panu Pitkänen is responsible for this chapter. The testing was done as a team with Eero Maasalmi, but all research and documentation is made by Panu Pitkänen.

### 2.1 Android and the Open Handset Alliance

Google, along with the Open Handset Alliance (OHA), launched the mobile operating system Android in the end of 2007 (DiMarzio 2008, 23.) The Open Handset Alliance is dedicated to “accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience” (Open Handset Alliance 2007a.) Nowadays the OHA is represented by 84 of the leading software and hardware companies, and mobile operators, for example Sprint, T-Mobile, Intel, Asus and Acer in addition to Google and others (Open Handset Alliance 2007a, Open Handset Alliance 2007b.)

As the name of the Open Handset Alliance implies, the companies representing it are dedicated to contribute to the openness of the mobile world. This is why Android is an open source system bringing cheaper and more innovative products for customers and better developing platforms for programmers. As soon as Android was first released, development tools and tutorials were made available to help new developers find the new system. (Open Handset Alliance 2007a.)

The Open Handset Alliance has an advantage over other operating system developers, because Android as such is a ready package to be converted into different devices. For example, if LG Electronics would bring Android into some other device than mobile phones or tablets, the operating system would be ready and there would immediately be an uncountable amount of ready applications for the new devices.

## 2.2 Android from a technical perspective

Android is a stack of software built for mobile devices. It consists of an operating system, middleware and some key applications. Android applications can be programmed with the Java-language. (Android Developers 2011a.)

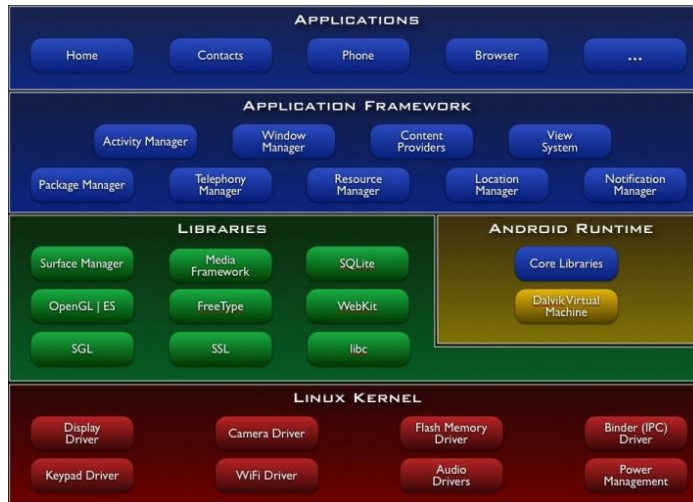


Image 1 - The major components of the Android OS (Android Developers 2011a.)

### 2.2.1 Applications

By default, all Android devices have an email client, a text messaging program, a calendar, a map software (usually Google Maps), an Internet browser, an application for managing contacts, and other core applications installed in them. More applications for Android can be downloaded from the Android Market; some of the applications there are free and some cost a fairly low amount of money. (Android Developers 2011a.)

### 2.2.2 Application Framework

Because of Android's open development platform, developers have free hands to build very innovative applications. It is possible for developers to take advantage of the device hardware, access location information, run background services, set alarms and add notifications to the status bar, basically anything that helps in creating the ideal application.

Being an open system, Android lets developers have full access to the same framework APIs, application programming interfaces, which are used by the core applications. (Android Developers 2011a.)

### **2.2.3 Libraries**

A set of C/C++ programming languages' libraries are included in Android. They are used by various components of the system and visible to developers through the application framework. Key libraries and their functions are described briefly below.

**System C Library:** a standard C system library tuned for embedded Linux-based devices.

**Media Libraries:** libraries supporting playback and recording of many popular image files and audio and video formats.

**Surface Manager:** unites 2D and 3D graphic layers from several applications.

**LibWebCore:** a modern web browser engine powering the Android browser and the embeddable WebView described later.

**SGL:** the main 2D graphics engine.

**3D libraries:** libraries using either hardware 3D acceleration or the 3D software rasterizer.

**FreeType:** used to render bitmap and vector fonts to the form needed.

**SQLite:** a lightweight database engine available for all applications to use.

(Android Developers 2011a.)

### **2.2.4 Android Runtime**

There is a set of core libraries in Android resembling the core libraries of the Java programming language and thus providing similar functionality (Android Developers 2011a.)

Android uses a certain virtual machine known as Dalvik to run each of its applications. These applications are run in their own processes each of which having their own instance of the Dalvik virtual machine. Thanks to the way Dalvik has been written, a

device can efficiently run multiple virtual machines at the same time. Applications are executed in the Dalvik Executable format in a way that minimal memory footprint is required. (Android Developers 2011a.)

Dalvik relies on the Linux kernel for matters concerning threading and low-level memory management (Android Developers 2011a.)

### **2.2.5 Linux Kernel**

The Linux kernel version 2.6 is the communicator between the device hardware and the software. The kernel also takes care of Android's core system services which include the overall security, memory management, process management, network related systems and the driver model. (Android Developers 2011a.)

## **2.3 Introduction to Android SDK**

The Android SDK provides a group of tools needed when developing applications for the Android operating system. It is recommended to use the Eclipse software development environment to run ADT, Android Development Tools, which then gives the user access to all of the SDK tools. (Android Developers 2011b.)

It is also possible to develop applications without Eclipse: this method requires the use of a text editor and the ability to use the SDK tools on the command line or with scripts. This is not such an easy way as using Eclipse but all the same tools are still available. (Android Developers 2011b.)

The most important SDK tools include android, which is an SDK and AVD Manager, the emulator and the Dalvik Debug Monitor Server, DDMS. In addition to these, there are also many other tools available and also included in the SDK starter package. (Android Developers 2011b, Android Developers 2011c.)

## **2.4 System and software requirements for developing with Android SDK**

The system and software requirements for developing applications with Android SDK are briefly listed in this section. For the original list, please see the Android developer web guide at <http://developer.android.com/sdk/requirements.html>.

### **2.4.1 Supported Operating Systems**

Android SDK runs on basically all commonly available operating systems. From Microsoft's operating systems the supported ones are the 32bit version of Windows XP, and both the 32bit and 64bit versions of Windows Vista and Windows 7. (Android Developers 2011d.)

Mac OS X 10.5.8 is the earliest version supported; only the 32bit versions are accepted. From the Linux side, Android SDK has been tested on Ubuntu and Lucid. The GNU C Library version 2.7 is the minimum requirement and if using Ubuntu Linux, version 8.04 is the earliest that will run Android SDK. If using a 64bit distribution, it must be capable of running 32bit applications. (Android Developers 2011d.)

### **2.4.2 Supported Development Environments**

The integrated development environment best capable of running the Android SDK is Eclipse IDE. From it you need the version 3.5 or higher. One of the following Eclipse IDE packages is recommended for Android development: Eclipse IDE for Java Developers, Eclipse Classic v.3.5.1 and higher, or Eclipse IDE for Java EE Developers. If not already included in the Eclipse IDE, it is necessary to install the appropriate JDT, Java Development Tools, plugin. (Android Developers 2011d.)

The JDK, Java Development Kit, version five or six is required. The JRE, Java Runtime Environment, alone is not enough. The installation of the Android Development Tools plugin is highly recommended. (Android Developers 2011d.)

### 2.4.3 Hardware requirements

In addition to the hard drive space required by everything listed before, you need to reserve at least 600 MB of free space on your hard drive for the recommended components needed to run the Android SDK properly. The more components are installed, the more space is needed. (Android Developers 2011d.)

Here is an example list of components and the space they require:

SDK Tools: 35 MB, SDK Platform-tools: 6MB, Android platform: 150 MB each, SDK Add-on: 100 MB each, USB Driver for Windows: 10 MB, Offline documentation: 250 MB, Samples: 10 MB per installed platform. (Android Developers 2011d.)

## 2.5 Installing the Android SDK

This section, along with each of the sections following, describes our own empirical research and findings related to installing the SDK, using it to implement applications and then publishing them. If any screen shot following has no source mentioned on it, it is then taken from the environment used in the context and is copyrighted by the developer of the appropriate environment.

After checking that the computers we intended to use the SDK with met the system requirements, we started the installing process. First, we navigated to the home page of everything related to Android development, <http://developer.android.com/index.html>. (Android Developers 2011e.) The Android SDK can be downloaded free of charge and the downloading instructions can be found from under the “SDK”-tab. On the first page, we downloaded the installing package according to the computer’s operating system. When using Microsoft Windows, it is recommended to download the .exe file. It is an automatic installer for the SDK, but downloading the .zip package works as well in the end. In the end of this site, there was a link to the installing instructions. (Android Developers 2011f.) While selecting the destination for all downloadable files throughout the whole installation process, we found one folder including everything downloaded to be quite handy.



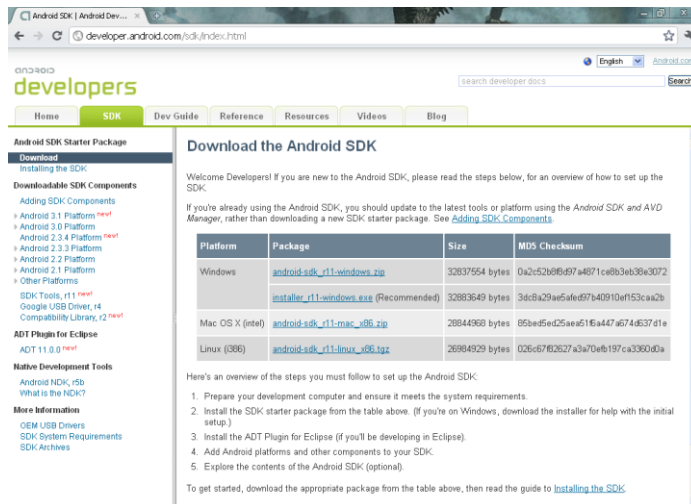


Image 2 - List of available Android SDK packages (Android Developers 2011f.)

The first step was to install the JDK, Java Development Kit from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. It is a necessary add-on to the Android SDK because the programming language used is Java. It was important to choose to download the JDK package as the JRE, Java Runtime Environment at itself is not enough. (Android Developers 2011g.)

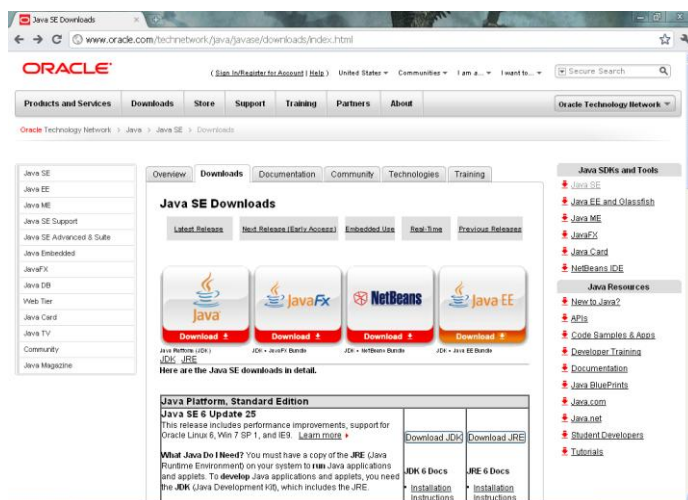


Image 3 - Java Standard Edition Downloads (Oracle 2011a.)

From under the “Downloads”-tab the license agreement needed to be accepted before choosing the right installation file according to the operating system.

Double-clicking the downloaded installation file starts the installer. The setup program first prepared the Installation Wizard.

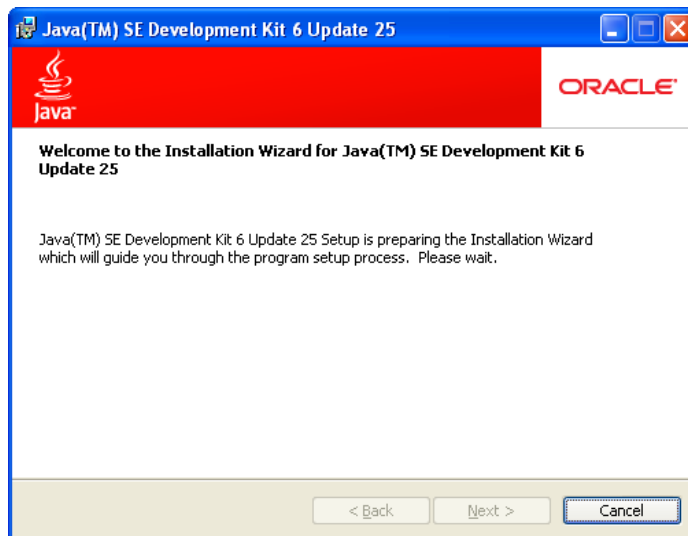


Image 4 - JDK Installer

When prompted, we pressed “Next” to carry on to the next step.

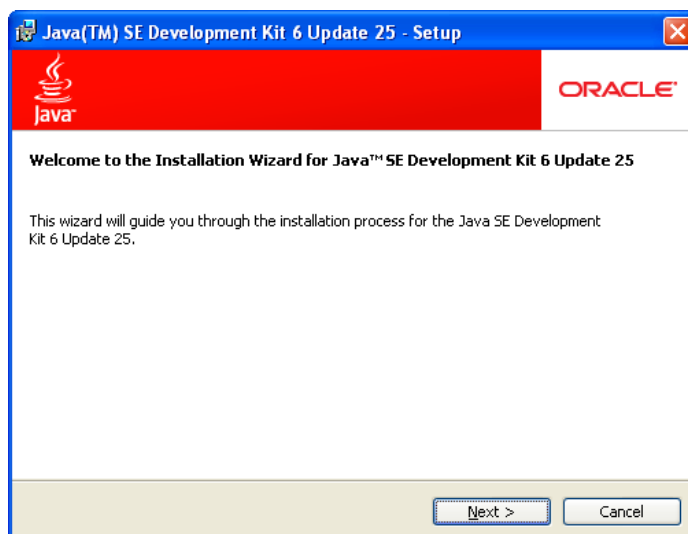


Image 5 - JDK Installer

It was possible to select the features of the JDK that we wanted installed. We found the best way to be just to leave the features list at its default setting. Here it is also possible to change the installation path if needed.

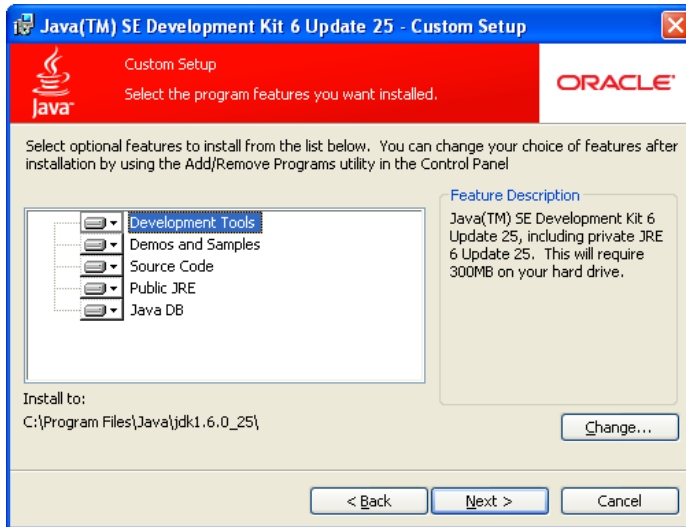


Image 6 - JDK Installer

In the next window after the installer had finished, the program still offered a chance to change the installation path. We left it at its default. Finally, the installer finished. After pressing the “Finish”-button, the system automatically opened an Internet web site for registering the JDK. Registering is not compulsory; it can be done at the user’s free will.

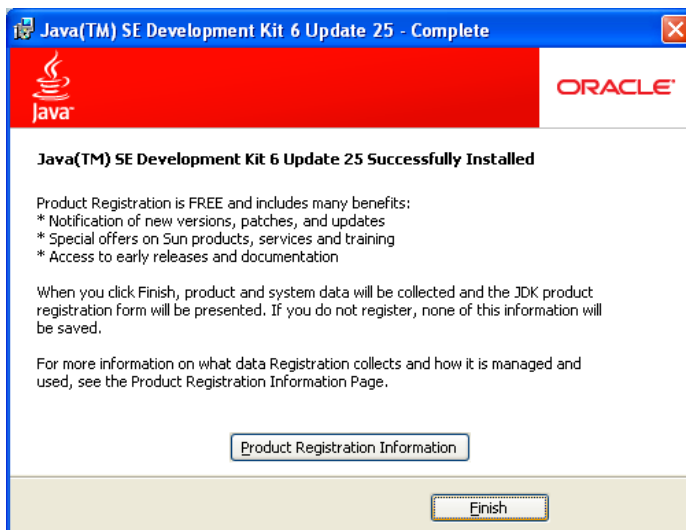


Image 7 - JDK Installer

Next, we went to the website <http://www.eclipse.org/downloads/> in order to install the Eclipse IDE. Eclipse is the software development environment into which the Android development tools plugin is later added. It is not compulsory to install Eclipse as other similar programs work too, but the installation of it, Eclipse Classic to be precise,

is strongly recommended. (Android Developers 2011g, The Eclipse Foundation 2011a.)

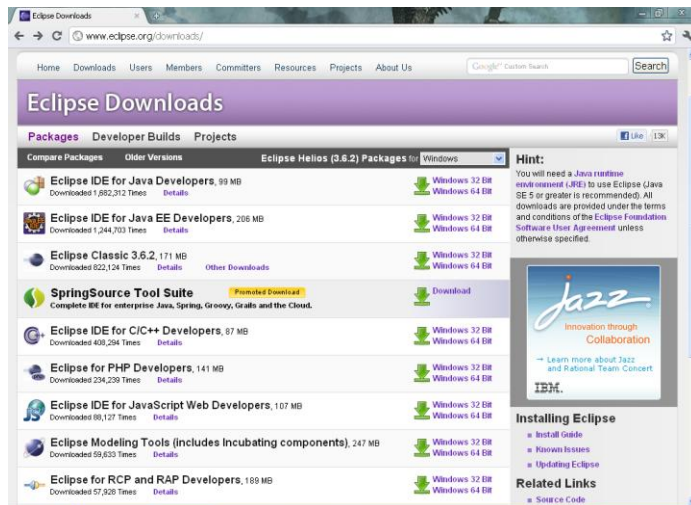


Image 8 - The Eclipse Downloads (The Eclipse Foundation 2011a.)

The newest version of Eclipse Classic and choosing the appropriate downloading link according to the operating system used ensure a successful installation. The site led us to a mirror from which we then downloaded the software.

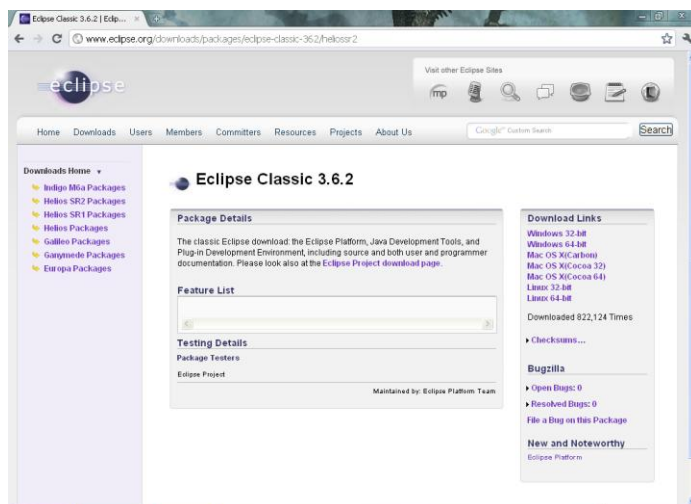
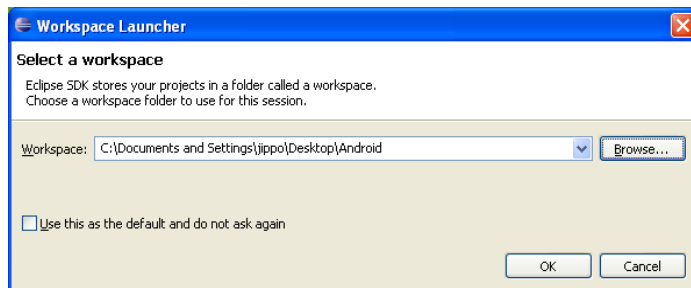


Image 9 - The Eclipse Downloads (The Eclipse Foundation 2011a.)

The first thing Eclipse asked after starting the download was the destination of the workspace folder. In this folder, it saves all the projects being formed with the SDK. After successful installation, the front page of Eclipse Classic opened and we moved on to the next step of the installation.



**Image 10 – Installing Eclipse SDK**

In the second part of the web guide, the instructions were quite unclear. Especially in our case as we downloaded the before mentioned .zip package of the Windows installer on one computer and the .exe package on another. By reading the web guide, at least we got the feeling that only unpacking the .zip folder would be enough at this phase. It isn't, the SDK needs to be installed along with at least one platform. Whereas downloading the .exe installer gave no other option than to install the SDK. (Android Developers 2011g.)

Either by searching inside the unpacked .zip folder and executing the application file from there, or by executing the downloaded .exe file, the setup for Android SDK Tools will begin.

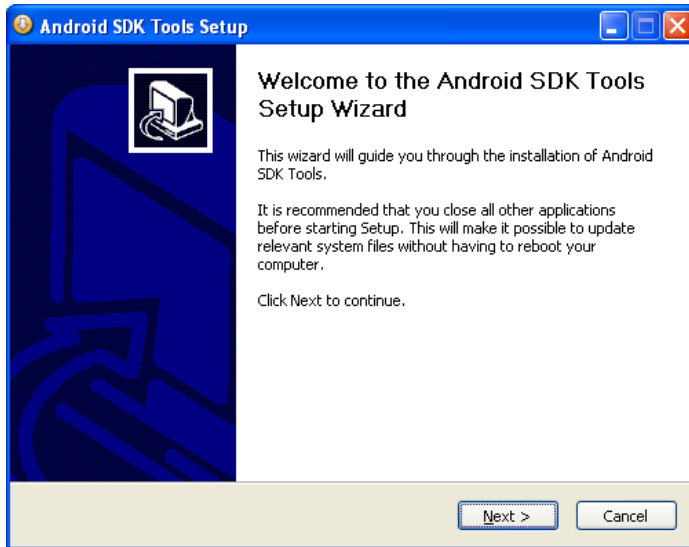


Image 11 - Installing Android SDK Tools

The executable version (.exe file) of the installer searches for the previously installed JDK from your computer. We ran this application on Windows 7 Professional 64bit and Windows XP Professional 32bit. On the computer running Windows 7, the installer did not find the JDK in this phase even though it was installed and even re-installed, and the computer was booted in between. The computer running Windows XP had no problem in finding the JDK with this installer.

We then executed the application file from the .zip package with the computer running Windows 7. That application skips this part and the installation continued onwards without problems.

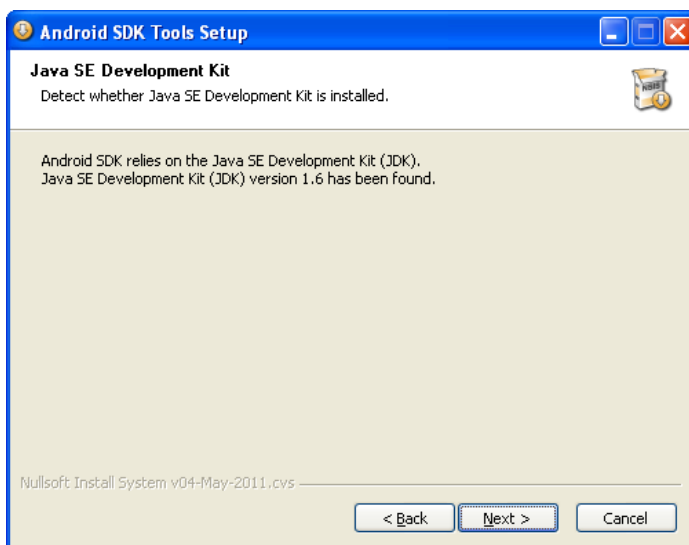


Image 12 - Installing Android SDK Tools

The installer asked for the installation path which we left at its default value. It also required us to give a folder menu for all the shortcuts to be installed into. We left that also at its default value.

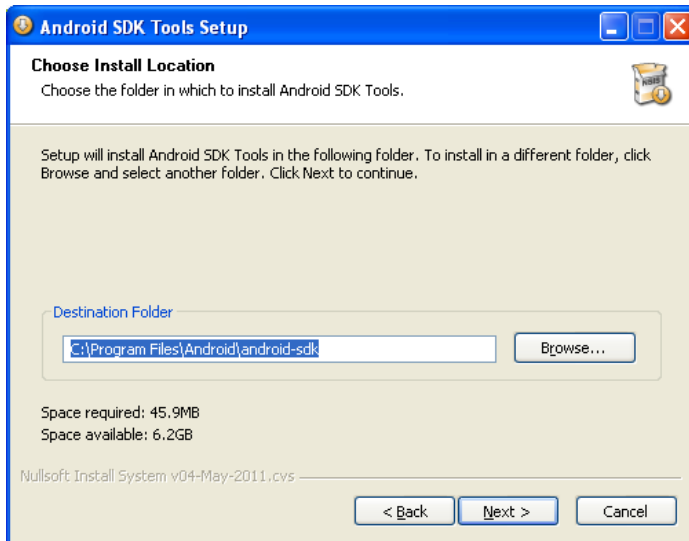


Image 13 - Installing Android SDK Tools

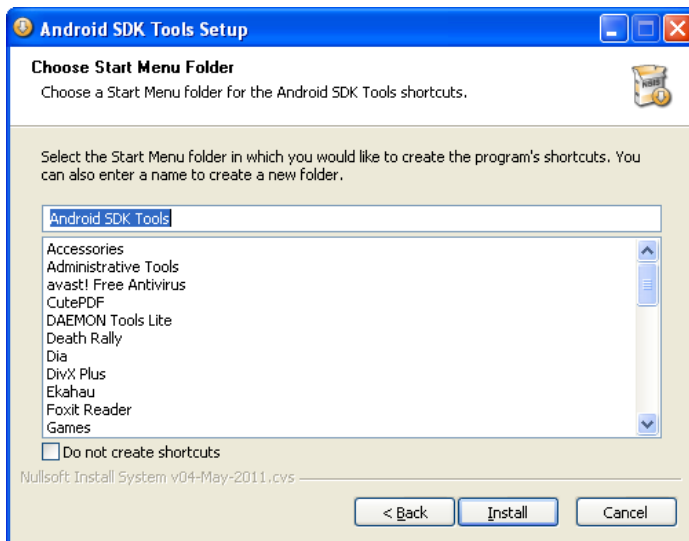


Image 14 - Installing Android SDK Tools

After all necessary choices and check-ups were made, the installation began.

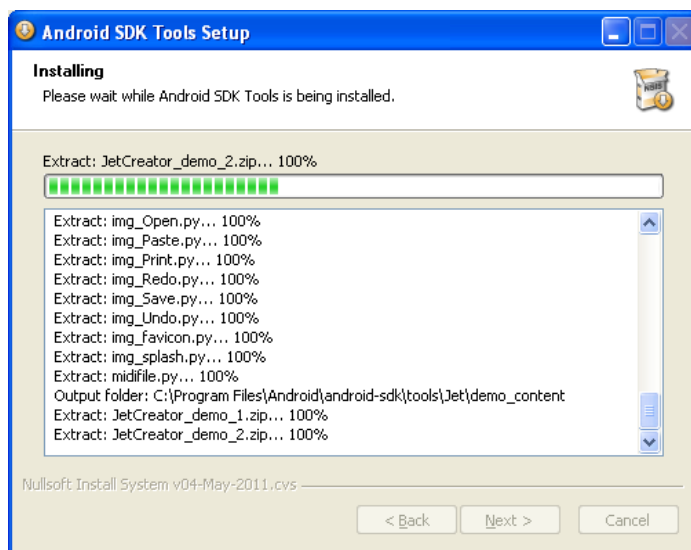


Image 15 - Installing Android SDK Tools

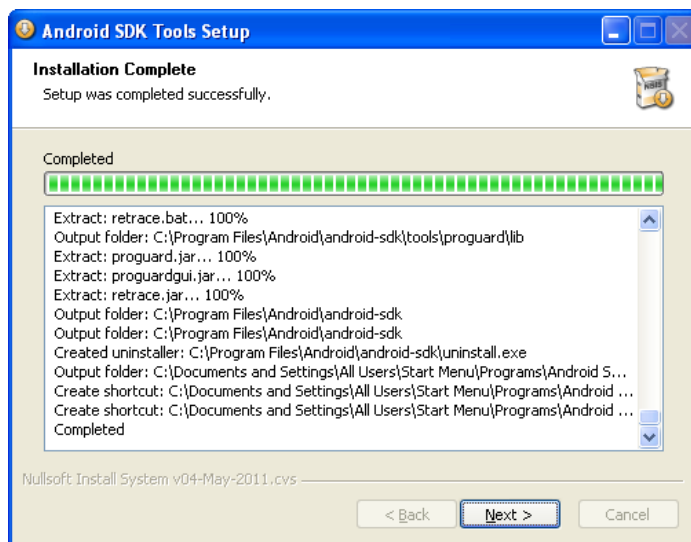


Image 16 - Installing Android SDK Tools

After the installation was done, we started the SDK Manager by checking the appropriate box and pressing “Finish.”

First, the SDK Manager refreshed its sources to find all available platforms to be installed.



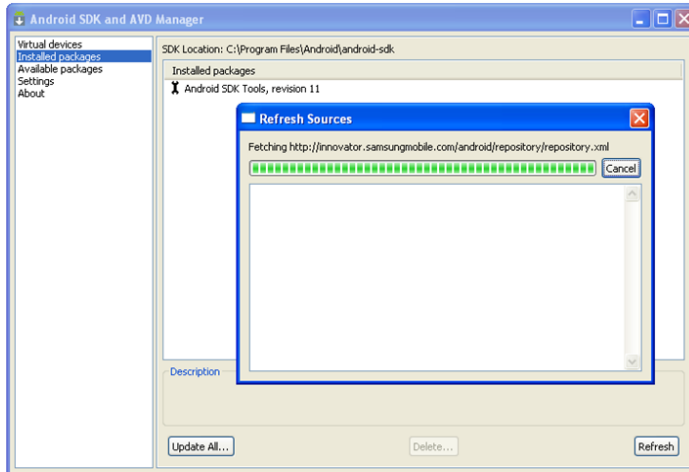


Image 17 - Starting Android SDK Manager

Next, we chose the platforms and tools to be installed according to the list of recommendations available in the Android Developer web guide (Android Developers 2011g). We chose the Basic and Recommended-layouts listed in the web guide. These included all SDK Tools, SDK Platform-Tools, SDK Platforms, Documentation, Samples and USB Drivers (for debugging on a mobile device) to be installed.

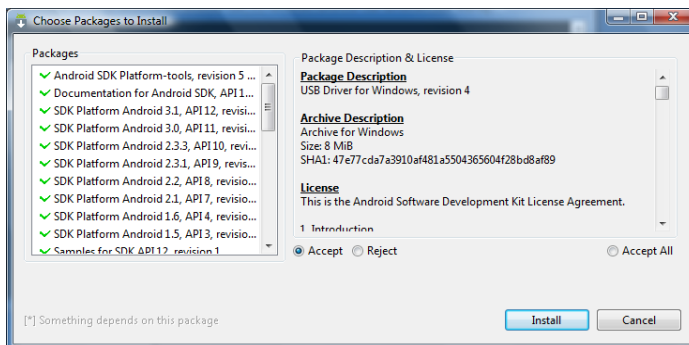


Image 18 - Setting up the Android SDK Manager

The platforms and tools were then installed; this took a while depending on the network connection speed.

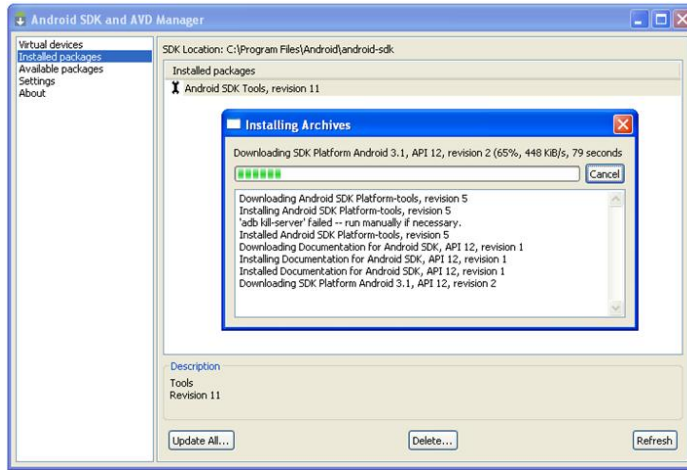


Image 19 - Setting up the Android SDK Manager

When the extra packages were installed, the installer asked whether we would like to restart ADB. We accepted this. When the installation window claimed that the installation was done, it could be closed. After this, all installed packages could be found from behind the “Installed packages”-tab on the Android SDK and AVD Manager main view.



Image 20 - ADB restart required after setting up the SDK Manager

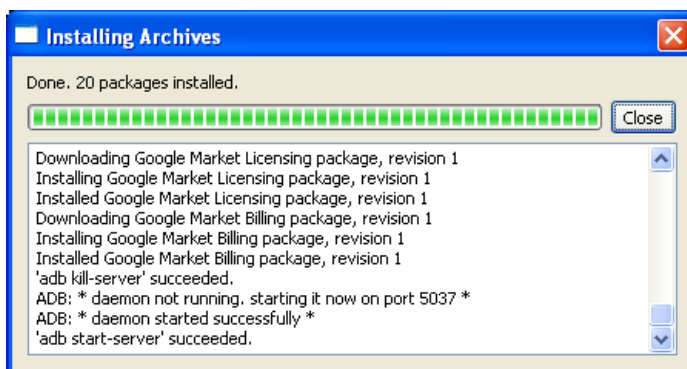


Image 21 - Updated archives after the installation

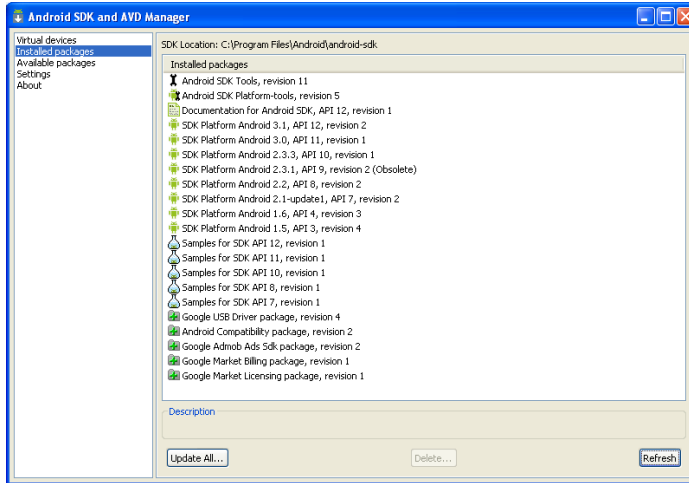


Image 22 - Installed packages listed

Next, we installed the ADT (Android Development Tools) Plugin for the previously installed Eclipse Classic.

The first step was to open Eclipse Classic from the path where it was chosen to be installed. The application can be started from the same folder where it was downloaded in the first place. Then, by navigating through Help > Install New Software, we got started. (Android Developers 2011g.)

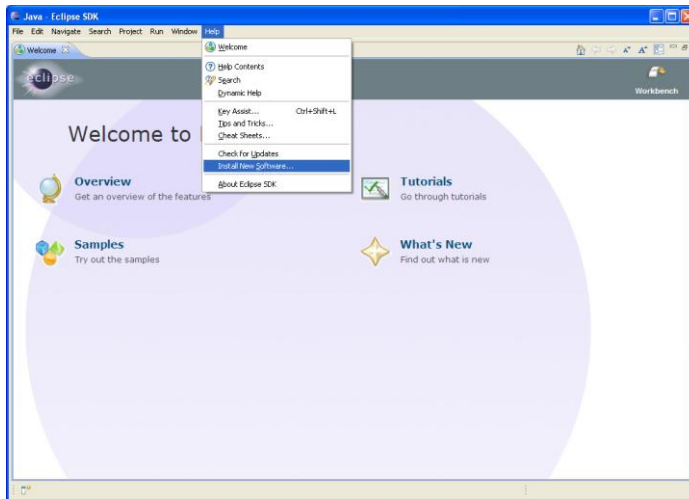
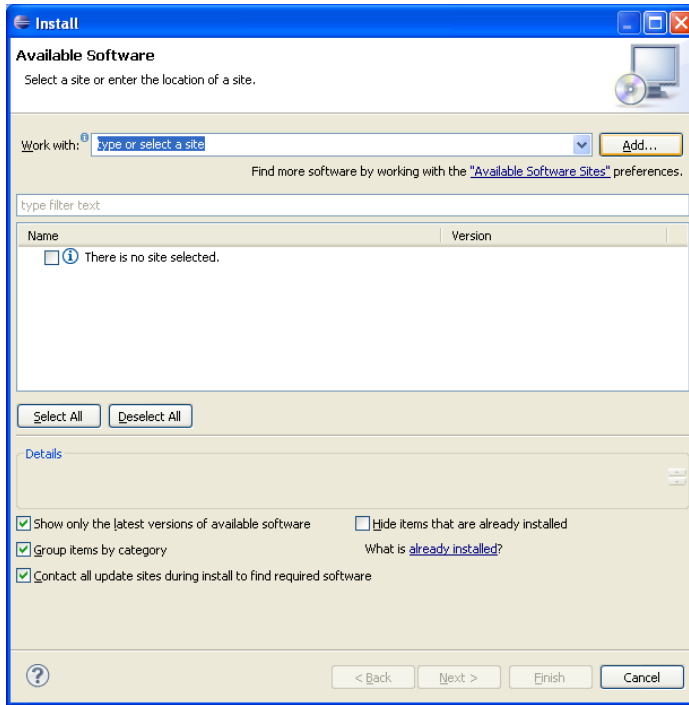


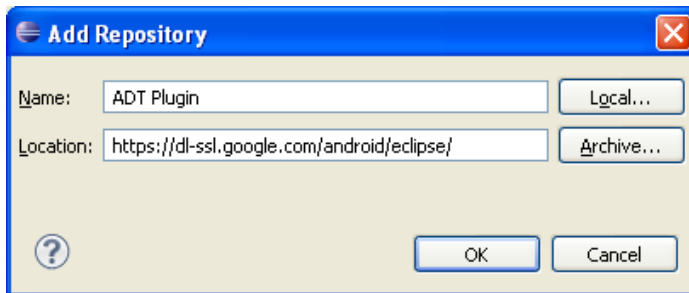
Image 23 - Installing Android Development Tools in Eclipse

When opened, the installer window looked like this. We pressed the “Add”-button from the top right corner to start the installation of the ADT Plugin.



**Image 24 - Installing Android Development Tools in Eclipse**

The Add Repository window opened next. Here, we wrote “ADT Plugin” as the name and “https://dl-ssl.google.com/android/eclipse/” as the location of the repository. We did not make these up or search for them; they were mentioned in the installation guide. (Android Developers 2011g.)



**Image 25 - Installing Android Development Tools in Eclipse**

After pressing "OK" in the previous window, we returned to the main installing view. We made sure that the checkbox next to "Developer Tools" was selected so that all four items would be installed. We then pressed "Next."

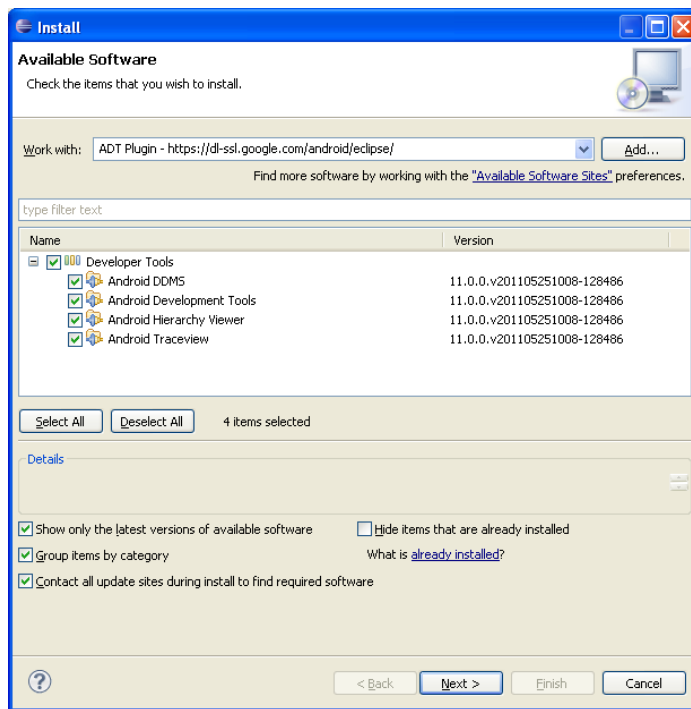


Image 26 - Installing Android Development Tools in Eclipse

The next window only reviews the items to be installed, we again pressed "Next."

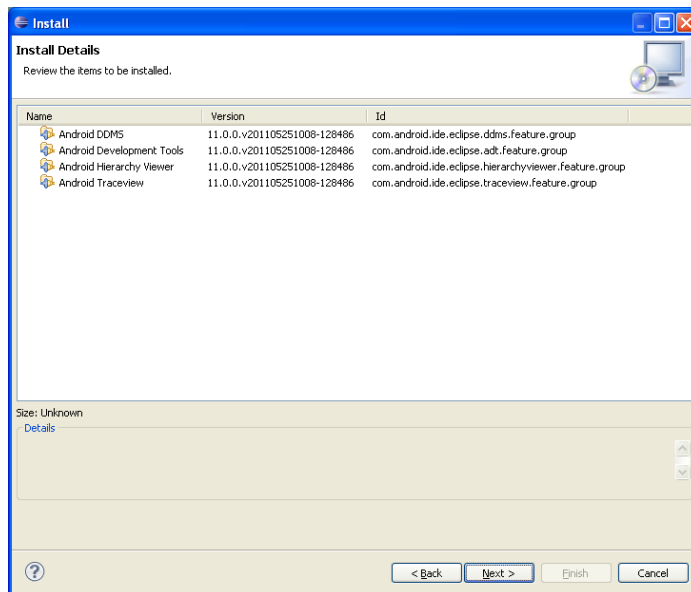


Image 27 - Installing Android Development Tools in Eclipse

After this, the licenses of the software to be installed needed to be read and accepted.  
After doing this, the installation began by pressing “Finish.”

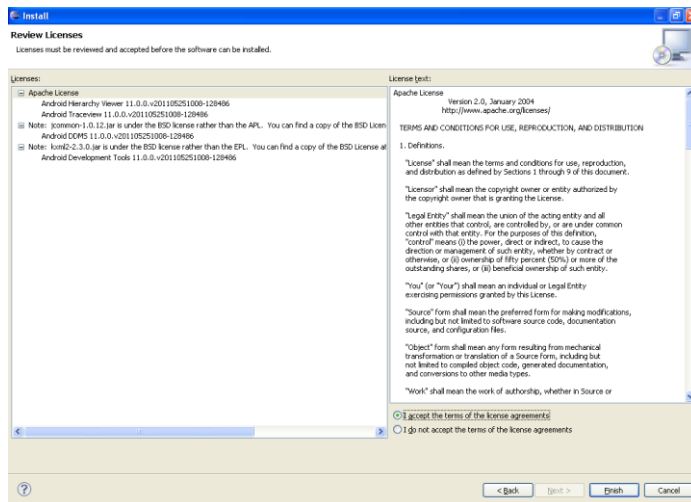


Image 28 - Installing Android Development Tools in Eclipse

The installation did not last long, only for a minute or two. A security warning was also established as some of the items we were installing contained unsigned content. As the Android Developers installation guide advised just to press “OK” here, we did not hesitate to do so. The last part was just to restart Eclipse and then the ADT Plugin was successfully installed.

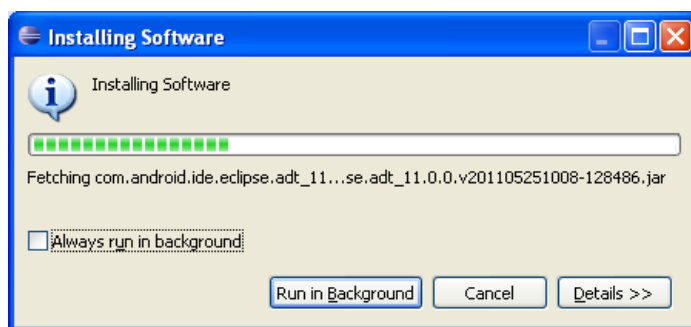


Image 29 - Installing Android Development Tools in Eclipse

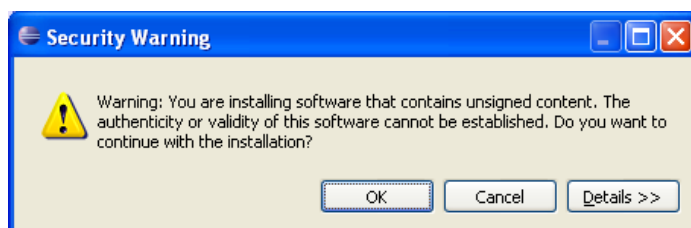


Image 30 - As instructed, it was safe to press OK here

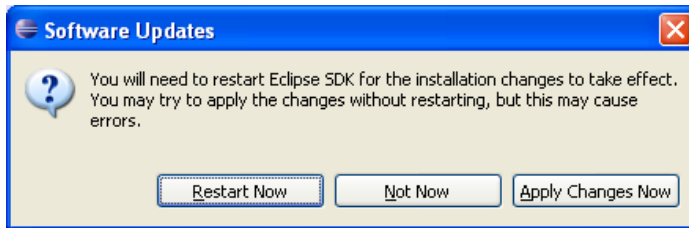


Image 31 - Installing Android Development Tools in Eclipse

After the plugin was installed, Eclipse still needed to be configured to use the previously installed Android SDK and its directory as a preference point for the ADT Plugin. This can be done from the main view in Eclipse and then navigating through Window > Preferences. (Android Developers 2011g.)

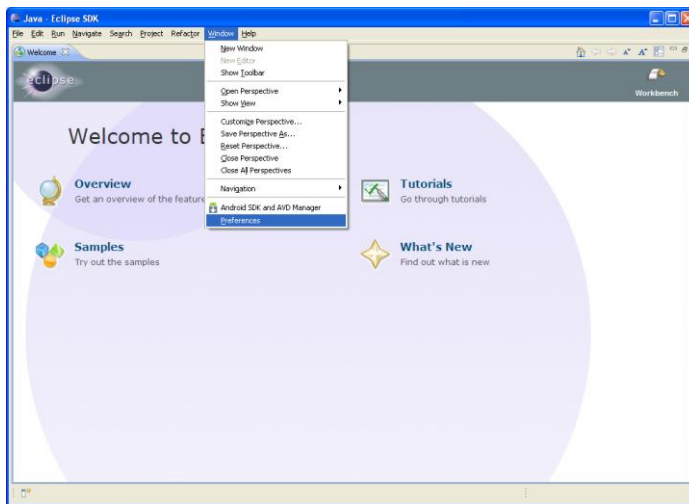


Image 32 - Configuring Eclipse to use Android SDK with the ADT plugin

The next window that appeared was about whether or not to send usage statistics to Google. This could be chosen by either checking or unchecking the box in the lower left side of the window and then clicking “Proceed.”

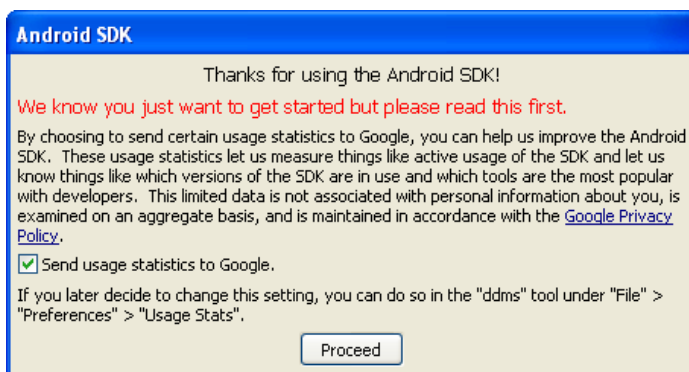


Image 33 - Configuring Eclipse to use Android SDK with the ADT plugin

Next, the Preferences window opened. By navigating to the Android tab from the right side of the window, the Android Preferences view opened. Here, we browsed the SDK Location to be the one into which the Android SDK was installed in the very beginning of the whole installation phase.

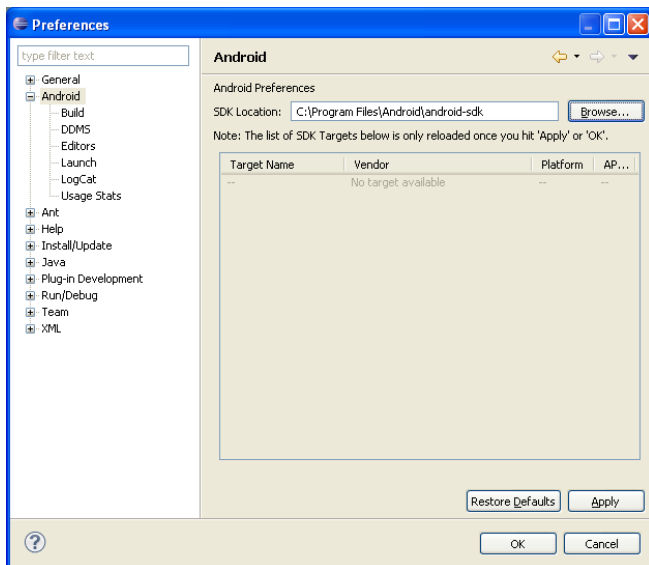


Image 34 - Configuring Eclipse to use Android SDK with the ADT plugin

The system then showed all the previously installed, now available, platforms for Android SDK. These could be approved by pressing “Apply” and then the Android SDK running on Eclipse Classic was successfully up and running and ready for some programming.

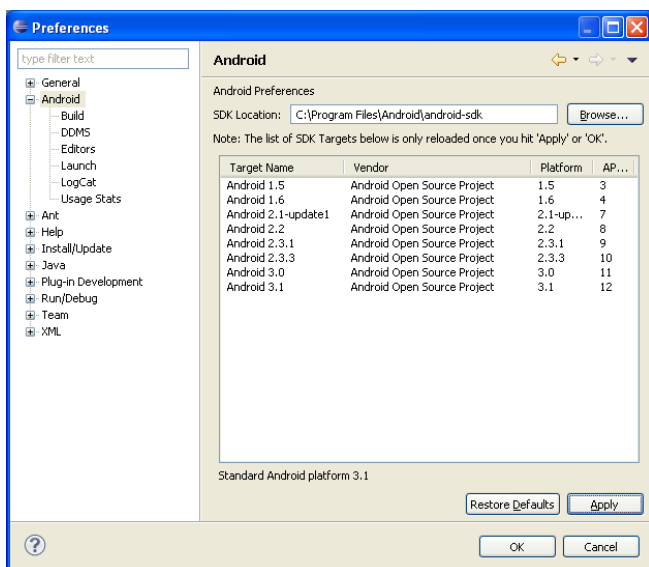


Image 35 - Configuring Eclipse to use Android SDK with the ADT plugin



## 2.6 Summary of the Android SDK installation process

Installing the Android SDK and Eclipse along with their add-ons was quite a complex task at first.

There were many different applications and add-ons to be installed and they had to be installed in a specific order, so without the instructions found from the website of Android Developers, the installation could not have been completed successfully. Even by following the instructions, we had to guess our way through a couple of steps.

On the other hand, while trying for the second time on another computer, the installation was much easier to finish. Almost no instructions were needed as the installation process was clearer.

It is understandable that the installation process is not the simplest, and as later learned, is much more difficult than the installation of the iOS SDK. Apple does not offer the most recent version of the iOS SDK for everyone to download for free; which gives them the chance to create such a package that includes only the needed application and add-ons. But on the contrary, Android SDK being free and available for everyone, they need to gather the needed environments and tools from separate, free sources.

Nevertheless, as a whole, the installation process of the Android SDK is no more complicated than installation process of any other commercial software. All the options are by default chosen so that the user has no need to change anything and can still have a perfectly functional SDK. As always, there is of course the custom install option, but it is mainly for those who need extra functionality or want to drop out an add-in they already know they will never need. Or like in this case, maybe someone wants to use another software development environment instead of using Eclipse. In such case, the developer is usually already an advanced user and doesn't necessarily even need the instructions for installing the environment.

## 2.7 Programming with the Android SDK

This section handles the part of developing applications using the Android SDK. The environment used here was built using the instructions given before. In order to follow these instructions and gain from our experiences, it is necessary to use the Eclipse software development environment along with the correctly installed Android SDK.

The Android Developers website, which can be found on

<http://developer.android.com/resources/tutorials/hello-world.html>, has been used as a help throughout this section, it is also mentioned as a source separately wherever necessary.

### 2.7.1 Setting up the Android Virtual Device

Before we could start implementing for Android, we needed to set up at least one Android Virtual Device (AVD.) The AVD is basically the emulator used when testing the software; it consists of all the environment settings of the given version of Android. In this example we built our AVD to use Android 2.3.3, the most common Android platform at the moment.

First, after opening Eclipse Classic, we navigated to Window > Android > Android SDK and AVD Manager. (Android Developers 2011h.)

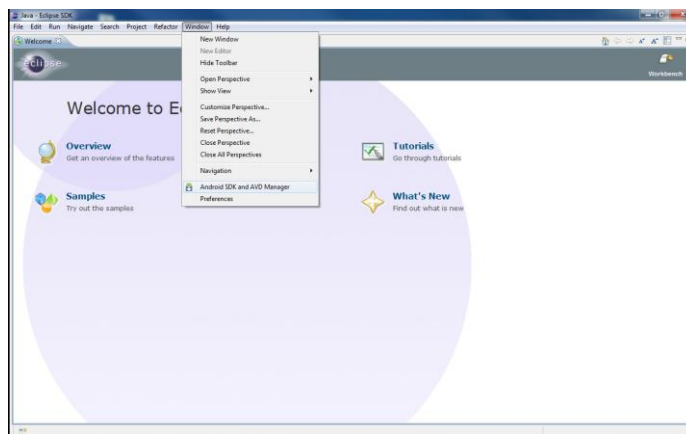


Image 36 - Setting up the AVD

From the Android SDK and AVD Manager, we clicked “New” to start creating the virtual device.

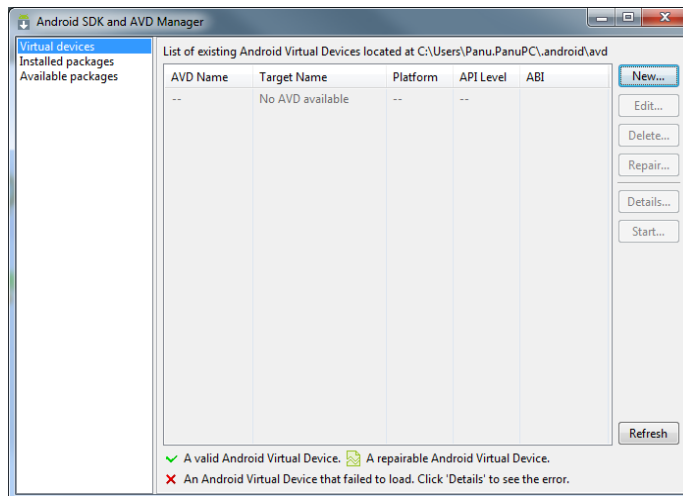


Image 37 - Setting up the AVD

In the next window, the AVD name and target needed to be given. The target is the Android platform required for the emulator to run. In this example we chose Android 2.3.3 – API Level 10 as it is the most popular platform at the moment. (Android Developers 2011h.)

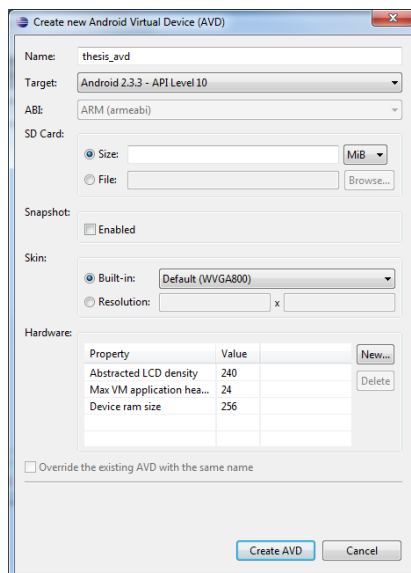


Image 38 - Setting up the AVD

Now, the AVD was created and the next step was to start a new application project in Eclipse.

## 2.7.2 Starting the project in the Eclipse environment

In the Eclipse main view, we navigated to File > New > Project in order to start building a new implementation project. (Android Developers 2011h.)

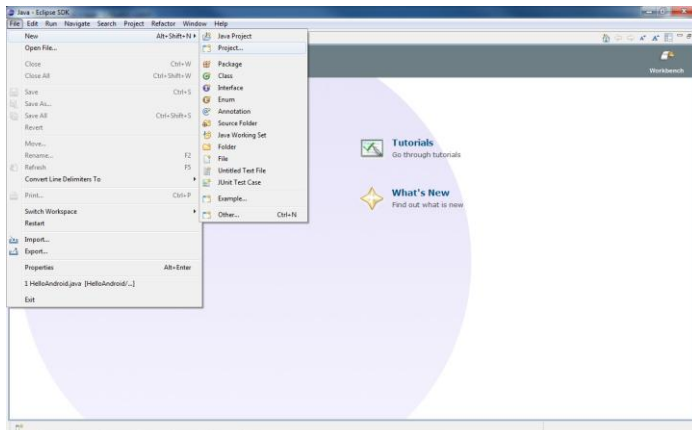


Image 39 - Starting a project

We navigated to Android > Android Project and pressed Next.

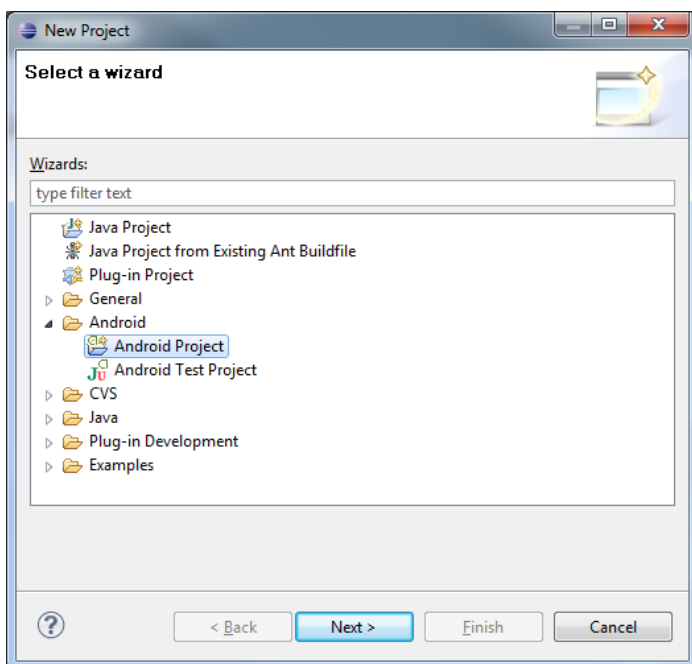


Image 40 - Starting a project

In the following window, Eclipse requests for the key information about the project. Unless everything required is given, the project cannot be started. The project name needs to be given before anything else. The build target is the target environment into

which the application will be created. This can be at the most the same platform as the AVD created earlier is running, but it can also be an earlier one. In fact, Android platforms are forwards compatible, so if the created application works on Android 1.6, it will also work on Android 2.3.3. The same is not necessary true in the opposite direction; an application created with the build target of Android 2.3.3 might not work on systems running Android 1.6 for example. (Android Developers 2011h.) This is why we chose our project to be built into the Android 1.6 platform. When scrolling down the New Android Project-window, the Application name and Package name still need to be given.

In our example, the filled fields are as following:

Project name: HelloWorld

Build Target: Android 1.6

Application name: HelloWorld

Package name: com.example.helloworld

Create Activity: HelloWorldActivity

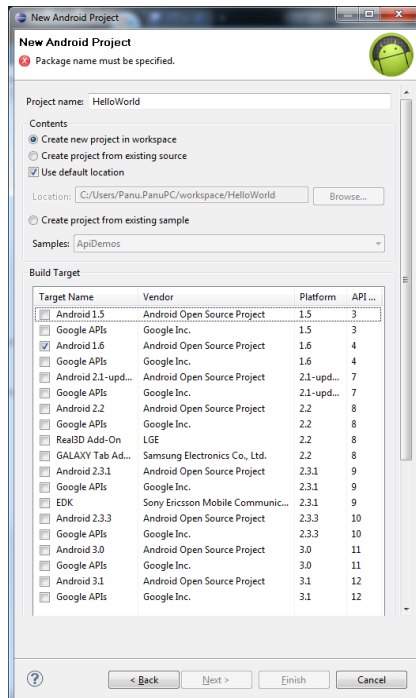


Image 41 - Starting a project

Now, the project was created and everything was ready for application implementation.

### 2.7.3 Developing HelloWorld for Android

As already seen in the steps taken before, the first application we created with the Android SDK was a simple Hello World-application. The application we created displays a button on the screen and after pressing the button, it displays a pop-up message.

Neither of us have prior experience in the Java programming language, so this implementation was a useful lesson to us. As mentioned before, the Android Developers website was a great help in this part also as it showed the basics of the development language and also the differences between implementing the application directly into the source code and using the XML implementing possibility offered by Eclipse Classic (Android Developers 2011h.) The end result is the same, but with XML, the implementation is done more graphically than within the source code.

After learning the basics, we created an application with the instructions given by Android Code Monkey (Greg Zimmers.) His implementation uses the XML-format and after a few tries, was easy to build and get working. (Android Code Monkey 2010.)

Here are screen shots of the application we implemented in use with the AVD emulator. The source code for the application is found in Appendix A.1. - HelloWorld for Android. The application was debugged and seemed to work without errors.

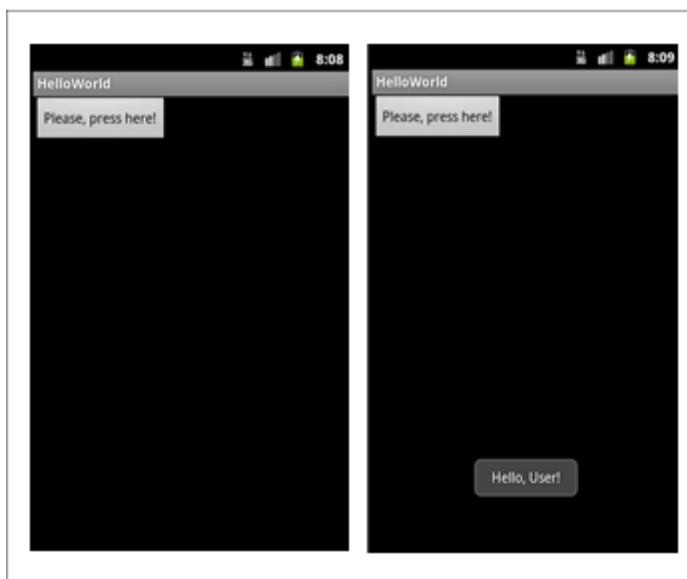


Image 42 - HelloWorld for Android

## 2.7.4 Developing OpenSomeWebsite for Android

The second application we created opens the Finnish television program guide <http://www.telkku.com> in the device's browser after the user presses a button on the application. It is easy to change the site to be opened; it is only one line in the source code. We made this application with source code combined from the previously used Android Code Monkey's HelloWorld and Android Developer's WebView introduction. (Android Code Monkey 2010, Android Developers 2011i.) The source code for the application is found in Appendix A.2. - OpenSomeWebsite for Android. We used a combination of XML and traditional source code when implementing this application.

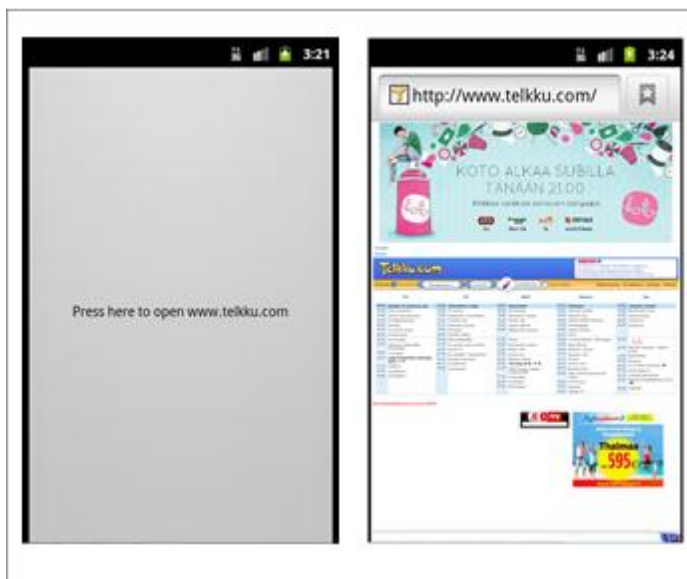


Image 43 - The website as it opened in the application (Kustannusosakeyhtiö Iltalehti 2011.)

This application opens the device's designated Internet browser. Android supports another way of viewing websites also, it is called WebView. We tried it, as a default it only opens the site requested and if some link is pressed on that site, the real web browser automatically opens instead. This can be overridden with appropriate JavaScript code, but we did not find it worth the trouble. WebView is a good way to demonstrate a static website in the middle of an application working, but it is no web browser. (Android Developers 2011j.)

### **2.7.5 Summary of the application development process on Android SDK**

Developing applications for Android most definitely is not a hard process after some practicing. Java is a very popular programming language, so the Internet is full of tutorials for it, which means that it is possible to create software even without being very familiar with the language. After a short while of getting familiar with the system, Eclipse is a very user friendly environment to work in.

XML seemed to be a nice addition to programming for Android with Eclipse. It was easier to create graphical objects, such as buttons, with XML when their source code was immediately embedded into the file after inserting the object. This made it easier to focus on the outcome of the actual application, not needing to waste energy on the correct implementation of objects.

On the negative side, the AVD emulator on Android SDK works quite slowly. It takes a while to start, at first we thought it crashed during start-up. When it starts, it works quite sluggishly. For a person not being very fluent in Java, the language indeed occasionally brings up problems, but nothing that could not be researched and learned from.

## **2.8 Deploying an application on Android**

In this section, we will demonstrate how to prepare the HelloWorld application built before for publishing and how to distribute the readymade .apk-package to other people also. In this example all steps reported until now have been taken and default options were used while installing the SDK. All instructions for the publishing process can be found from the Android Developers website.

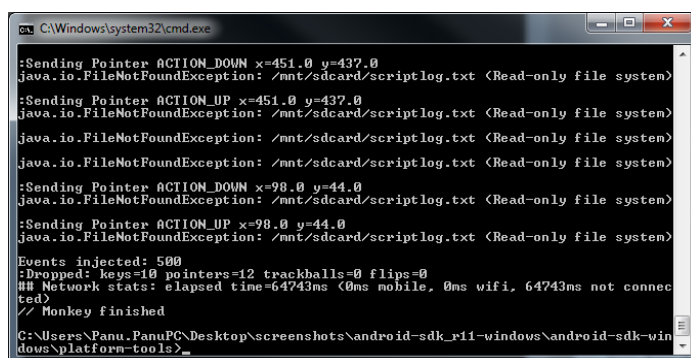
It is important to be familiar with using the command prompt of the computer's operating system. If using a Windows system and/or a user account with decreased rights, it is necessary to run the command prompt as an administrator.



## 2.8.1 Testing the application on Android SDK

First, the application needs to be tested. The more complex the application, the more thorough tests are necessary. Simple usability tests can be done on the emulator while developing, but test cases and stress tests are also recommendable. We tested the application on the emulator and then used the UI/Application Exerciser Monkey included in the Android SDK to generate random user events and system stress events. As a result, our application did not crash or get mixed up in any way. (Android Developers 2011k, Android Developers 2011l.)

The UI/Application Exerciser Monkey is used via the command prompt. First, we had to navigate to the folder where we unpacked the Android SDK-files mentioned in the beginning of section 2.5. There, the needed path was `\android-sdk-windows\platform-tools`. This folder includes a tool known as ADB, Android Debug Bridge, which is needed to run the stress test. After navigating to this path with the command prompt, we opened Eclipse Classic and our project, and ran it in the emulator. Then we returned to the command prompt and inserted the following commands: “adb shell monkey -p com.example.helloworld -v 500.” Com.example.helloworld is the name of our package; it is of course substituted by the name of the current project in question. This command sends 500 pseudo-random commands to the application. (Android Developers 2011l.) The test ended without error messages.



```
C:\Windows\system32\cmd.exe
:Sending Pointer ACTION_DOWN x=451.0 y=437.0
java.io.FileNotFoundException: /mnt/sdcard/scriptlog.txt (Read-only file system)
:Sending Pointer ACTION_UP x=451.0 y=437.0
java.io.FileNotFoundException: /mnt/sdcard/scriptlog.txt (Read-only file system)
java.io.FileNotFoundException: /mnt/sdcard/scriptlog.txt (Read-only file system)
java.io.FileNotFoundException: /mnt/sdcard/scriptlog.txt (Read-only file system)
:Sending Pointer ACTION_DOWN x=98.0 y=44.0
java.io.FileNotFoundException: /mnt/sdcard/scriptlog.txt (Read-only file system)
:Sending Pointer ACTION_UP x=98.0 y=44.0
java.io.FileNotFoundException: /mnt/sdcard/scriptlog.txt (Read-only file system)
Events injected: 500
Dropped: keys=10 pointers=12 trackballs=0 flips=0
## Network stats: elapsed time=64743ms (0ms mobile, 0ms wifi, 64743ms not connected)
// Monkey Finished
C:\Users\Panu.PanuPC\Desktop\screenshots\android-sdk_r11-windows\android-sdk-windows\platform-tools>
```

Image 44 - Using the UI/Application Exerciser Monkey

A list of other possible commands and their combinations can be found from the Android Developers website,

<http://developer.android.com/guide/developing/tools/monkey.html>.

## **2.8.2 Considering the use of license agreements**

It is worth considering whether the application needs an End User License Agreement (EULA) provided in order to protect the implementer's person, organization, and intellectual property. (Android Developers 2011k.)

If the coming application will be released with a cost through the Android Market, it is smart, yet fully optional, to add support for Android Market Licensing. If needed, this gives the implementer control of the application after the user has purchased it. Using Android Market Licensing makes the application query Android Market at run time to check their licensing situation. This makes it possible to disallow further use of the application making it a strong weapon against piracy among other advantages. (Android Developers 2011m.)

## **2.8.3 Giving the application an icon, a label and a correct version number**

In order to get the application verified and published, it is necessary to add an icon, a label, a version code and version name for the application.

In Eclipse Classic this could all be done very easily from the Android Manifest. First, we opened our HelloWorld application in the Package Explorer on the left side of the screen. Then, we double-clicked `AndroidManifest.xml` from the bottom of the list in order to open the Manifest. (Android Developers 2011k.)

The label and icon could be inserted into the appropriate fields in the "Application"-tab of the Android Manifest. The label is basically the name of the application showing in the main menu of the user's mobile device. As a default, the system finds the label from the application resources located in the path `res > values > strings.xml > app_name`. This same name is also used elsewhere in the application, so if no real reason appears for altering it, it should be left as it is. (Android Developers 2011k.)

The icon is the picture of the application showing in the menu of the user's mobile device. As a default, the icon of our application was a picture of the Android mascot.

We created a 36x36 pixel .png-format picture of a smiling face and drag-and-dropped it into the drawable-ldpi folder in the application resources. This could then be found by browsing the appropriate text box in the Android Manifest.

It is important to notice that there are three different drawable-folders in the system resources: hdpi, mdpi and ldpi. “Dpi” stands for dots per inch and the preceding letter stands for either high, medium or low. The default icon sizes in the folders were ldpi 36x36px, mdpi 48x48px and hdpi 72x72px. This should be taken into consideration when building applications for several Android platforms and screen sizes. The smallest icons understandably will not look very good on a 10 inch tablet device and the biggest will not show correctly on a 2.5 inch mobile device.

The version code and name could be altered from the ”Manifest”-tab of the Android Manifest. In the Manifest, there were two fields: Version code and Version name. Version code is the version number visible only for other applications and systems. It needs to be an integer and thus, according to general recommendations, we used the number 1 to represent our application’s first development phase (Android Developers 2011k.).

The Version name is the version number visible to users. It can be a decimal and as our application is in its first phase, we used 0.1 to represent it.

Here are screen shots of both of the tabs we completed in our Android Manifest in this step:

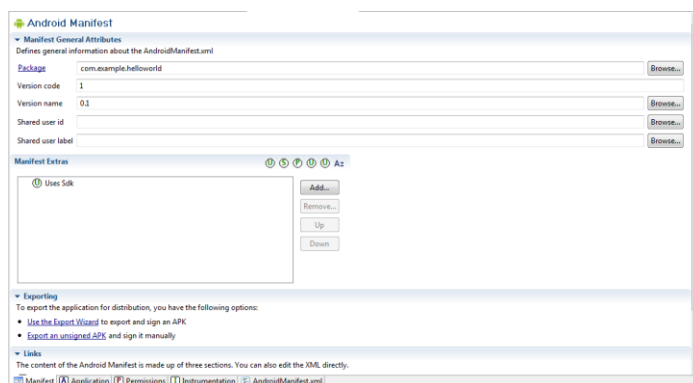


Image 45 - Android Manifest, General Attributes

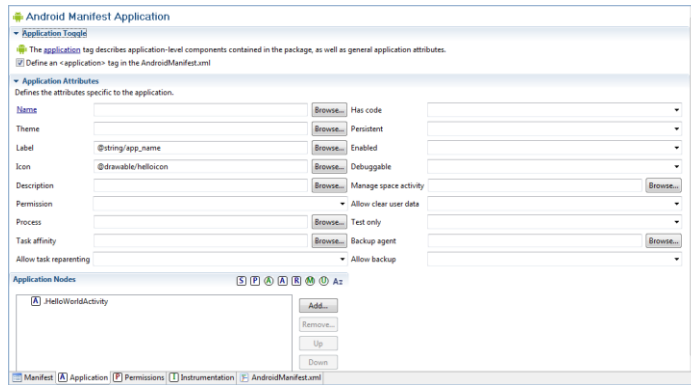


Image 46 - Android Manifest, Application Attributes

## 2.8.4 Turning off unnecessary facilities and removing unnecessary files

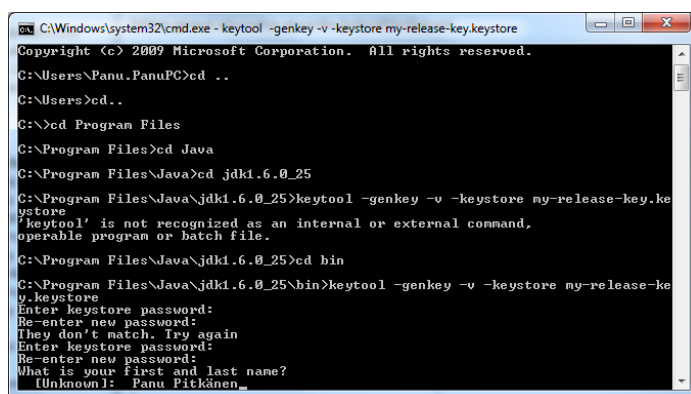
It is important to make sure that debugging and logging facilities are turned off before compiling the application. This can be done simply by removing private data, log files, backup files and other files you know to be unnecessary from the application project. We also checked that the “Debuggable”-textbox from the “Application”-tab of the Android Manifest does not show it to be turned on.

## 2.8.5 Generating a cryptographic key and considering the use of Maps API Key

The most demanding part of compiling the application was the use of the cryptographic key. The use of it, however, was required, so this step required some focus. The key was built with a tool known as Keytool, which came along in the JDK tools.

First, we opened the Windows command prompt with administrative privileges. Then we used it to navigate to the path where the JDK was installed earlier. If default installation choices were used, the path required is: C:\Program Files\Java\jdk1.6.0\_25\bin, of course depending on the version of the installed JDK. After navigating to this path with the Command prompt, we inserted the following command to generate the required keystore as a file called “my-release-key.keystore”: “keytool -genkey -v -keystore my-release-key.keystore -alias alias\_name -keyalg RSA -keysize 2048 -validity 10000” (Android Developers 2011n).

After running the command, Keytool required passwords for the keystore and key. These were the only parts that needed answering, everything else could be skipped by pressing the Enter button, or you could have also answered the parts you wished. The command created the cryptographic key into a keystore-file located in the folder we were working in with the command prompt. After running the command, the key is valid for 10 000 days, which is about 27 years. Android Market requires the key to be valid until 22 October 2033 at the minimum (Android Developers 2011n). Before the application can be signed with the key, it needs to be compiled.



```
C:\Windows\system32\cmd.exe - keytool -genkey -v -keystore my-release-key.keystore
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Panu.PanuPC>cd ..
C:\Users>cd ..
C:\>cd Program Files
C:\Program Files>cd Java
C:\Program Files\Java>cd jdk1.6.0_25
C:\Program Files\Java\jdk1.6.0_25>keytool -genkey -v -keystore my-release-key.keystore
'keytool' is not recognized as an internal or external command,
operable program or batch file.
C:\Program Files\Java\jdk1.6.0_25>cd bin
C:\Program Files\Java\jdk1.6.0_25\bin>keytool -genkey -v -keystore my-release-key.keystore
Enter keystore password:
Re-enter new password:
They don't match. Try again
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Panu Pitkänen
```

Image 47 - Creating the cryptographic key

If the developed application uses Google Maps for any action, the application needs to consist of a MAPS API Key. This can be asked for from Google after reading their documentation about the matter: <http://code.google.com/intl/fi-FI/android/add-ons/google-apis/mapkey.html> (on 29.9.2011.) This requires concurring to their license agreement after which a MD5 fingerprint is formed. The application will be signed with this fingerprint and then a MAPS API KEY is provided by the Maps registration service. The Key must be referred to whenever using the MapView in the source code. (Google Code 2011.)

## 2.8.6 Compiling, signing and aligning the application

Our application needed to be compiled in release mode in order for it to be released for users. This was a simple step and it could be done in Eclipse.

First, we right-clicked our project in the Package Explorer and selected Android Tools > Export Unsigned Application Package (Android Developers 2011h).

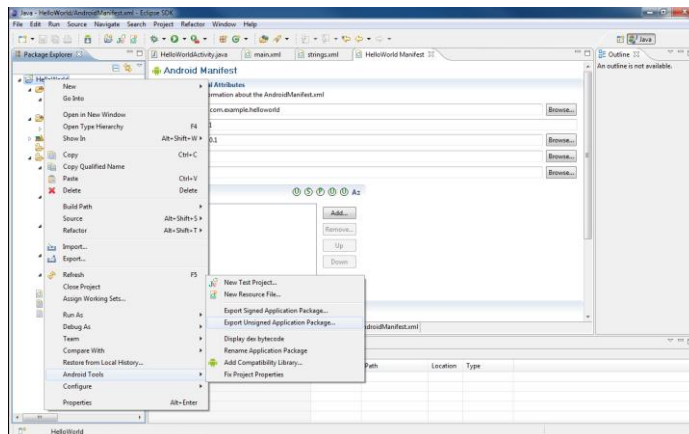


Image 48 - Exporting the Application Package

When choosing the destination for the unsigned application, it made our work easier to choose a simple path, because we still needed to access the application via the command prompt.

The signing of the package was done with a tool called Jarsigner and the cryptographic key we generated earlier. Again, with the command prompt running administrative privileges, we navigated to the location of Jarsigner in the Java-tools. In our case it was located in the same directory as Keytool: C:\Program Files\Java\jdk1.6.0\_25\bin. There, we inserted the following command to sign the application with the key in the keystore located in the same directory as the Jarsigner-tool. Our unsigned application was located on our desktop, but to make the command simpler, we do not mention it here. It is easy to copy the location of a file on your Windows computer: just go to the file properties and copy and paste the path under Location. (Android Developers 2011n.)

```
jarsigner -verbose -keystore my-release-key.keystore path of the application\HelloWorld.apk alias_name (Android Developers 2011n.)
```

After providing the passwords required, our application was signed successfully. This can also be verified with the command: `jarsigner -verify HelloWorld.apk`. If the prompt prints “jar verified”, everything has gone as it should. (Android Developers 2011n.)

The final step was to align the final application package. This was done with a tool known as `zipalign` found from the Android SDK-tools. This ensured the alignment of uncompressed data bytes and optimizes performance to its best. We opened the command prompt with administrative privileges and navigated to the location of our Android SDK-folder and from there onwards to the “tools”-subfolder. In there we printed the following command: `zipalign -v 4 path of the application\HelloWorld.apk HelloWorld.apk`. This did not take long and we also received a notice of successful alignment.

### **2.8.7 Releasing the application into the Android Market**

Our application was now finished and compiled successfully. It was a good idea still to test it now, as it was ready for publishing.

Before publishing an application on Android Market, the developer needs to register with the service using a Google account. The first steps are to create the personal developer profile, pay a registration fee of \$25.00 (on 29.9.2011) and agree to the terms of service. After this, it is possible to upload the application, update it and publish it when ready. Once the application has been published, it is possible for users to see it, download it and rate it. (Android Developers 2011o.)

Requirements without which the Android Market will not accept the application for distribution:

1. The application must be signed with a private cryptographic key which is valid until 22 October 2033 at least.
2. The application must have both the Version code and Version name defined.
3. The application must have the icon and label defined.

(Android Developers 2011o.)

## 2.9 Legal issues and costs of setting up an Android SDK environment

One of our objectives was to find out how much it would cost to run the Android SDK in the Eclipse environment in a classroom of 30 computers. It was assumed that the classroom already has the computers installed, so the only costs we paid attention to were the license fees related to the separate programs.

From the very beginning, Android was designed to be an open-source software stack with no liabilities to any industry players (Android Open Source Project 2011.) The source codes for all different Android releases are available in <http://source.android.com/>. The Android SDK can therefore be freely used in educational use – to create, test and publish software for free or with a fee. The only cost in developing software for Android is the \$25.00 when registering to become a developer in Android Market. Publishing in Android Market is not compulsory, however. At school the readymade applications can be uploaded to the school's own servers usually for free.

Java follows the GNU General Public License (later referred to as GNU GPL) (Free Software Foundation 2006.) Unlike normal software licenses, the GNU GPL guarantees your freedom to share the free software and receive the necessary source codes if needed. Software created following the GNU GPL can be distributed with a fee, but the end user must have the rights to share the software onwards as the license states. (Free Software Foundation 2007.)

In the case of using the necessary JDK alongside the Android SDK, no payments whatsoever are needed. A direct quote from the related paragraph in the Java license agreement in Oracle's website states the following:

SOFTWARE INTERNAL USE FOR DEVELOPMENT LICENSE  
GRANT. Subject to the terms and conditions of this Agreement and restrictions and exceptions set forth in the README File incorporated herein by reference, including, but not limited to the Java Technology Restrictions of these Supplemental Terms, Oracle grants you a non-exclusive, non-



transferable, limited license without fees to reproduce internally and use internally the Software complete and unmodified for the purpose of designing, developing, and testing your Programs. (Oracle 2011b.)

This means that the JDK can be used for free in educational, non-commercial use, inside the school.

Eclipse itself is written in Java, so its distribution is done according to the before mentioned GNU GPL. It may be used freely wherever, the license agreement states the following:

Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form (The Eclipse Foundation 2011b.)

Considering that it is not compulsory to upload files to Android Market and since all the necessary development kits are free, and assuming that there already exists 30 working computers in a school's IT-laboratory, the only costs from using the Android SDK build up from purchasing the required amount of mobile devices needed to test the software. In any case, these devices are not even necessary; the SDK includes its own emulator.

## 3 iOS

Eero Maasalmi is responsible for this chapter. The testing was done as a team with Panu Pitkänen, but all research and documentation is made by Eero Maasalmi.

### 3.1 iOS in general

iOS (formerly known as iPhone OS) is a UNIX-based operating system built for Apple's iPhone, iPod touch and iPad mobile devices. It is used for managing the hardware of a device and for providing technologies required to implement both native and web applications. Not many people know that IOS is originally Cisco's core operating system which has been in use for almost 20 years. Cisco licensed the trademark "iOS" to Apple in March 2010. Cisco's technology was not included in the license. (Carvell 2010.)

iOS was first introduced and released as the operating system of iPhone when on 29<sup>th</sup> of June 2007. 270,000 iPhones were sold in first two days and more than a million units in month and a half. (Trebitowski, Allen & Appelcline 2001, 2.)

At first there was no native SDK available. Apple claimed that there is no need for one, and that only JavaScript, CSS and HTML should be used for building applications for the device. This meant that Apple was only supporting web application development and did not see the need to support native application development. Even though Apple had locked the iPhone from the developers, only a few months after the release, the open source community had gained access, reverse-engineered the SDK and built a tool allowing the development of native applications for the device. This led to popularity of "jail breaking" the device so that it was possible to run third-party applications on it. Finally, in March 2008 Apple changed its mind and released the iOS SDK for the public. (Allan 2010, 1.)

### 3.2 iOS from a technical perspective

iOS can be presented as a proxy between the hardware of the hardware and the applications that appear on the device's screen. The applications rarely interact with the hardware directly. Instead they interact through system interfaces which protect the application from hardware changes. (Apple 2010a.)

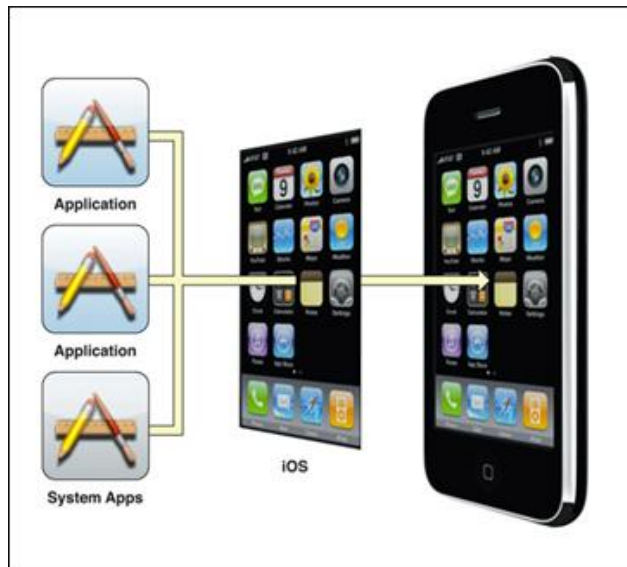


Image 49 – Different layers of the iPhone: Applications, iOS, hardware

iOS applications are developed using an object-oriented language called Objective-C, which supports the same basic syntax as C. Objective-C adds Smalltalk-style messaging to C-language. Smalltalk was one of the first object-oriented programming languages. (Apple 2010b.)

Even though iOS was designed for mobile devices, it shares a variant of the same basic Mach kernel which is used in Mac OS X, and many technologies with it. On top of the kernel, iOS has four technology layers: Core OS, Core Services, Media and Cocoa Touch. Lower layers consist of core services and technologies, and the higher level layers consist of more advanced and sophisticated ones. The layers all have different functionality. (Apple 2010c.)

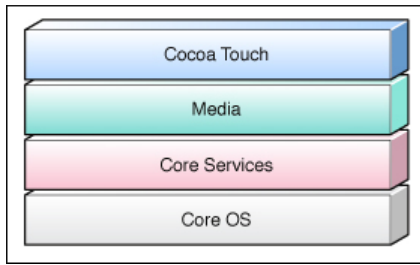


Image 50 – iOS Technology Layers

### 3.2.1 Core OS and Core Services layers

Core OS and Core Services are lower level layers. The features in these layers are often services which are not used directly in application, but which other frameworks use. These layers include interfaces for accessing files, low-level data types and network sockets. (Apple 2010a.)

Key technologies of Core OS layer include for example system frameworks for handling file-system access, memory allocation and mathematical computations, and security framework for data securing purposes. Core Services include SQLite for embedding SQL databases in applications, and XML support. Interfaces in both layers are mostly based on C programming language. (Apple 2010a.)

### 3.2.2 Media layer

On upper layers the technologies tend to have a bit more advanced features. Media layer contains graphics, audio and video technologies. AirPlay is also built in this layer. It allows cordless streaming of audio to Apple TV and speakers that support AirPlay. (Apple 2010a.)

Important features like support for animation, 2D and 3D rendering, video and still image manipulation are part of graphics technologies. Audio frameworks include for example easy access to user's iTunes library and audio playback and recording. Last, the media technologies consist of features like video recording and playback. The technologies in the Media layer are mostly C-based, but it also contains an advanced Objective-C based animation engine called Core Animation. (Apple 2010a.)

### **3.2.3 Cocoa Touch layer**

Every iOS application uses the UIKit framework of Cocoa Touch layer to implement such basic features like cut, copy and paste, battery state information, support for text and web content, and user interface -, and application management. For this reason the Cocoa Touch Layer is often called the home of the key frameworks for building iOS applications. Most of its technologies use the Objective-C language. (Apple 2010a.)

Gestures like tapping, pinching, swiping etc. are part of this layer's touch-based input technology. Another nice feature called the push notification lies in this layer. Push notification alerts the user of new information even when the application is not running. (Apple 2010a.)

### **3.3 iOS Developer Programs**

Apple offers four different Developer Programs for iOS developers to choose from. Being a part of a program not only has benefits, but is also compulsory if one for example wants to have access to Apple Developer Forums, submit Technical Support Incidents, or deploy applications on iPad, iPhone and iPod touch. The programs differ from each other by price and by benefits. (Apple 2010e.)

Apple Developer is the so called base program. If the developer wants to be a part of any iOS program, he first has to register as an Apple Developer. It is free, but only offers the access to the Dev Center Resources and the iOS SDK. (Apple 2010e.)

University Program is also free, but is only available for educational institutes and is of course meant for educational purposes. The University Program allows access to the Developer Forums and enables testing applications on real devices. (Apple 2010e.)

iOS Developer Program is a paid program with a \$99 yearly fee. Members of this program can submit Technical Support Incidents and are allowed App Store Distribution for their applications. There are both Individual and Company iOS Developer Pro-

grams, but the only difference between these two is that the Company Program allows the creation of Developer Teams. (Apple 2010e.)

Last there is the iOS Enterprise program for \$299/year. It has all the benefits the other programs have, and is mainly useful for those organizations that want to take advantage of the In-House Distribution of applications. In-House Distribution allows distribution of software for the organization’s members and employees. (Apple 2010e.)

	<b>iOS Developer Program - Individual</b> <a href="#">Learn more</a>	<b>iOS Developer Program - Company</b> <a href="#">Learn more</a>	<b>iOS Developer Enterprise Program</b> <a href="#">Learn more</a>	<b>iOS Developer University Program</b> <a href="#">Learn more</a>	<b>Registered as an Apple Developer</b> <a href="#">Learn more</a>
Dev Center Resources	✓	✓	✓	✓	✓
iOS SDK	✓	✓	✓	✓	✓
Select Pre-Release Software & Tools	✓	✓	✓		
Ability to Create Development Team		✓	✓	✓	
Apple Developer Forums	✓	✓	✓	✓	
Technical Support Incidents	2 per membership year	2 per membership year	2 per membership year		
Test on iPad, iPhone, and iPod touch	✓	✓	✓	✓	
Ad Hoc Distribution	✓	✓	✓		
In-House Distribution			✓		
App Store Distribution	✓	✓			
Price	\$99/year	\$99/year	\$299/year	Free	Free

**Table 1 – Comparing iOS Developer Programs**

### 3.4 Introduction to iOS SDK

The iOS SDK contains all the necessary tools for designing, creating, debugging and optimizing software for iOS. The list includes native - and web applications, but no other types of code like drivers, frameworks or dynamic libraries. Native applications appear on the device’s home screen and can be run without a network connection locally on the device. Web applications use HTML, CSS and JavaScript code and are located on a web server, are transmitted over the network and run inside a browser which – in the iPhone – is the mobile version of the Safari browser. (Apple 2010a.)

iOS SDK has built in frameworks, standard shared libraries, Xcode Tools (which includes Xcode, Interface Builder and Instruments), iOS Simulator and iOS Developer

Library. Xcode is the main application for developing applications and it is used for managing application projects, editing, compiling, running, and debugging code. Interface Builder is a tool for assembling the UI and Instruments is a tool for performing runtime analysis and debugging. iOS Simulator can be used for testing iOS applications by simulating the iOS technology stack on a Mac OS X. It makes testing faster, since the developer doesn't need to first upload the application to a mobile device. Even though a simulator is never the real system and thus isn't 100% reliable, it provides a nice and easy platform for testing. Last, the iOS Developer Library is the source for documentation helping application development. (Apple 2010a.)

iOS SDK is usually built in the Xcode package, but not always. This can be verified by checking it in the used installer's "Custom Install" pane. The most recent free version of Xcode at the moment (on 27<sup>th</sup> of June 2011) is version 3.2.6, which is available for those who are registered as Apple Developers. The most up-to-date version 4.0.2 can be purchased for \$4.99. Alternatively, members of the iOS or Mac Developer Programs can download 4.0.2 for free. (Apple 2010d.)

### **3.5 System and software requirements for developing with iOS SDK**

Xcode's software and hardware requirements are extremely trivial. All that is needed for the software to run is an Intel-based Mac running Mac OS X Snow Leopard version 10.6.6 or later and at least 10GB of free disk space. Snow Leopard (Mac OS X 10.6) was the first OS to support Intel-based products only, so basically all Mac computers manufactured after the release of Snow Leopard on 28<sup>th</sup> of August, 2009 should run Xcode.

There is an exception with the latest OS X (10.7 Lion) though. Since Xcode 3 is not compatible with it, Xcode 4.2 has also become available for free in the Mac App Store. (Apple 2009; Apple 2010-2011, Apple 2011a.)

### 3.6 Installing the iOS SDK

We began the installation of the iOS SDK by registering as an Apple Developer. It is not possible to download Xcode without an Apple ID which has been registered as an Apple Developer. After creating the account, we were able to download the Xcode and the iOS SDK from Apple's Developer pages: <http://developer.apple.com/xcode/>.

The images in this chapter are from the Mac OS X and the Xcode 3.2.6 Installer.

After downloading the 4.14GB “xcode\_3.2.6\_and\_ios\_sdk\_4.3.dmg” package, the file was shown in the Mac OS X's “Downloads” folder. These.dmg files are Apple disk image files and can be extracted by double-clicking the files.



Image 51 – Contents of OS X's "Downloads" folder after downloading the installation package

After the image file was extracted, two files were shown: “About Xcode and iOS SDK.pdf”, which is an introductory document to Xcode and iOS SDK, and “Xcode and iOS SDK.mpkg”, which is the actual installation file for Xcode and iOS SDK. We began the installation by double-clicking the “Xcode and iOS SDK.mpkg” -file. .mpkg files are Mac installer packages.



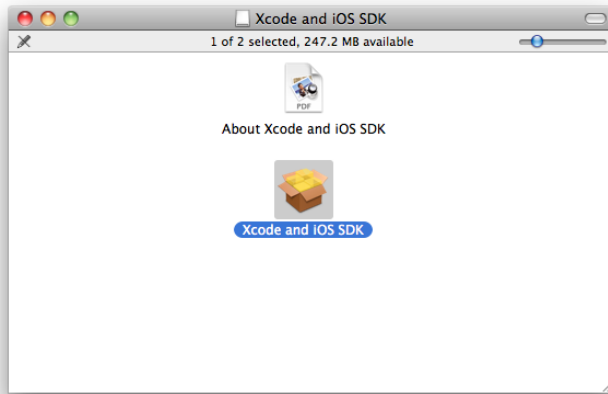


Image 52 – Contents of the “xcode\_3.2.6\_and\_ios\_sdk\_4.3.dmg” package

As soon as we began the installation, a security warning from the operating system itself appeared on the screen. It reminds the users to only install software from trusted sources. As this package was downloaded from Apple itself, we were safe to “Continue”.

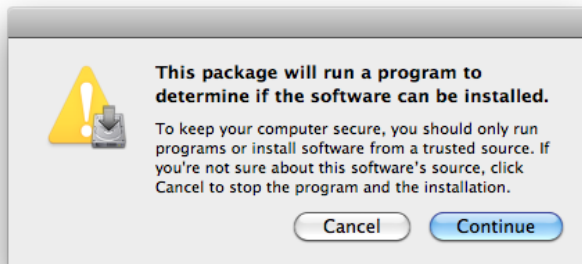


Image 53 – A security warning before beginning the Xcode and iOS SDK installation

The installation first displayed a short introduction screen welcoming us, the users, to the Xcode and iOS SDK installer. We selected “Continue” to read the license agreement.

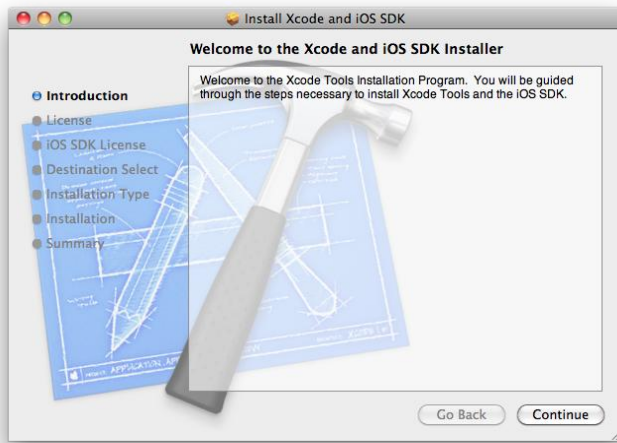


Image 54 – Xcode and iOS SDK installation software’s welcome message

Next, the Software License Agreement for Xcode was displayed. After reading the agreement, we again selected “Continue” to proceed with the installation.

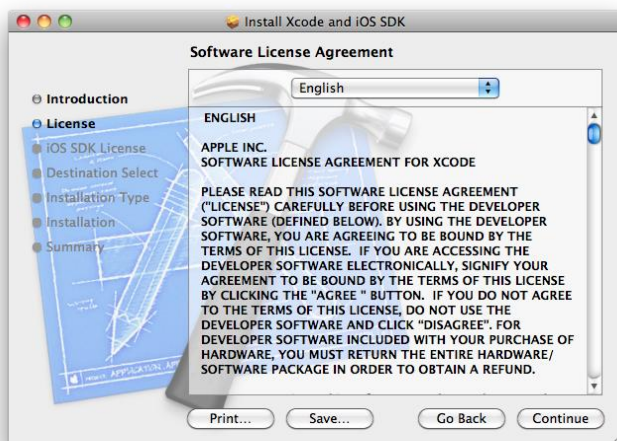


Image 55 – Software License Agreement for the Xcode

We agreed with the terms of the Xcode software license agreement and selected “Agree” to continue.

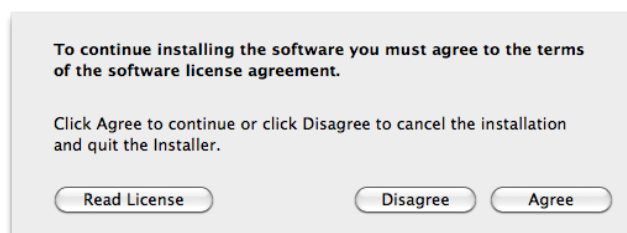


Image 56 – Agreeing with the terms of the Xcode

Another Software License Agreement was displayed. This time it was for the iOS SDK. We selected “Continue” to proceed with the installation.



Image 57 – Software License Agreement for the iOS SDK

We again agreed with the terms of the iOS SDK software license agreement and selected “Agree” to continue.



Image 58 – Agreeing with the terms of the iOS SDK Software License Agreement

Next the installer required us to select the install destination. The installer calculates the available space according to the destination. Our MacBook Pro only has one hard drive which was for that reason chosen automatically by the installer. We then selected “Continue” to go on to the next step.



**Image 59 – Selecting the install destination**

The Custom Install package selection pane was displayed next. This window shows all the packages available for installation. The installer displays additional information about what the packages include:

“Essentials”-package cannot be unselected, since it includes the complete Xcode development toolset: Xcode, Instruments, compilers and other tools. Both Mac OS X SDK and iOS SDK are installed by default, and iOS Simulator is also included. If the user doesn’t change the destination directory, “/Developer” folder is chosen by default. It should be noted here that the SDK developer tools will automatically replace any existing Apple development tools. (Apple 2010k.)

If selected, “System Tools” -package is also installed into “/Developer” folder by default. Only one installation, the most recent set of System Tools, per Mac OS X at a time is possible. The package includes system-wide tools like Shark, DTrace components for Instruments, and distributed build processes. (Apple 2010k.)

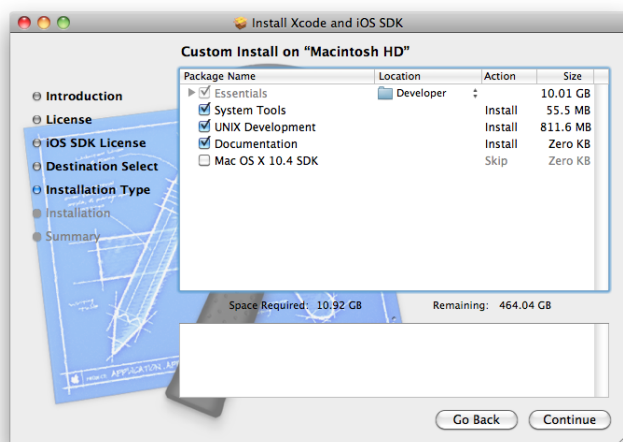
“UNIX Development” enables boot volume command-line development. This is done by automatically installing a clone of the GCC compiler and command line tools, which are part of the Xcode developer tool, into the boot volume. Other installed components are: header files, libraries, and other resources. It’s not possible to change the installation destination for this package. It is always installed onto the boot volume.

This package is mostly recommended for advance users who are more familiar with the SDK and can come by without the GUI, too. (Apple 2010k.)

“Documentation” enables offline documentation for Xcode. If this package is selected, Xcode automatically downloads developer documentation to disk at first launch and also keeps it updated. It is a good choice if the developer intends to work offline. (Apple 2010k.)

“Mac OS X 10.4 SDK” includes the support for Mac OS X 10.4 APIs development. There’s no use for it for iOS developers since it is used to develop Mac apps. (Apple 2010k.)

Since we were not interested in developing applications for Mac OS X, we used the default settings where the “Mac OS X 10.4 SDK”-package was already unselected. We then selected “Continue”.



**Image 60 – Selecting the components to be installed**

The installer then asked for a confirmation for the install location. After checking the location, we selected “Install” to finally begin the installation.

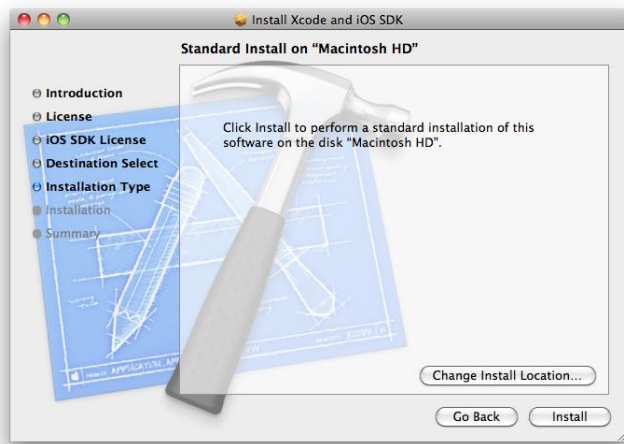


Image 61 – Confirming the install location

The installer asked for a user name and password before the installation was started. The user entered needs to have enough rights to write on the selected hard drive. We confirmed with “OK”.



Image 62 – Authentication is required for writing on the selected disk

After entering valid authentication information the installer began the installation. Even though the “Install time remaining” –calculator started the countdown from about 60 minutes, it soon came down, and in reality the installation with the MacBook Pro only took about 15 minutes in total.



Image 63 – Xcode and iOS SDK installation in progress

After running for about 15 minutes the installer notified us with a successful installation of Xcode and iOS SDK.

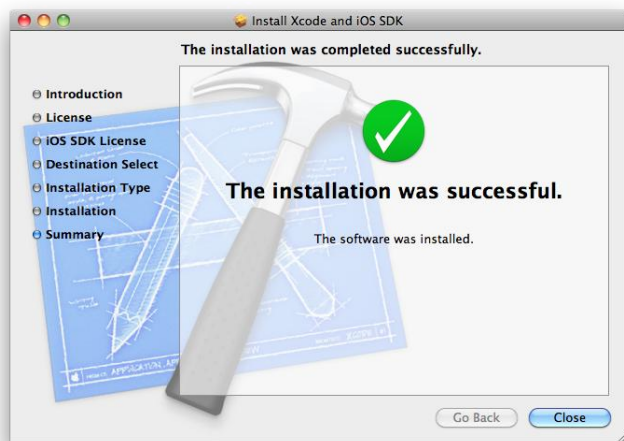


Image 64 – A message shown after a successful installation

### 3.7 Summary of the iOS SDK installation process

Installing the iOS SDK was a very straightforward process. Since one single installation file includes all the components needed, there was practically no room for error. The user may choose which version of Xcode he or she wants to download from the Apple Developer Center, but that is all basically it. There was no need to download third party installations files, add-ons, plug-ins nor extensions.



During the actual installation, the only step where the user had to actually choose something was the custom install pane where the installer asked to pick the packages which are installed. If the user has any intentions to develop applications for Mac OS X, it's an easy decision to include the "Mac OS X 10.4 SDK" -package. Anyhow, basically, the easiest way to go is to keep the default settings and select "Next" every time.

Installation of the tool has been made easy, but there we need to criticize the version releasing policy of Xcode. Apple has never released a single incremental update for the tool, which means that every time an updated version of the tool has been released, it has included a copy of the entire suite. For example if the user has downloaded and installed Xcode 4.0 or 4.0.1, he cannot patch the existing version, but is always required to download the entire suite to update Xcode to its most recent version. It's has not been possible to only patch the old version which has already been installed, but to first download the whole suite and then start the installation process from the beginning. Even with today's high-speed broadband internet connections and fast computers, downloading and installing the minimum of 4.5GB package takes its time. The same policy applies with the latest available Xcode version, too. (Bucanek 2010, 2.)

### 3.8 Programming with the iOS SDK

Xcode shares the same kind of simple looking user interface as Apple products often do. The main window consists of the "Toolbar", "Groups & Files list", "Detail View" "Editor" and "Status bar". Editor area is where the actual code is implemented. (Apple 2011b.)

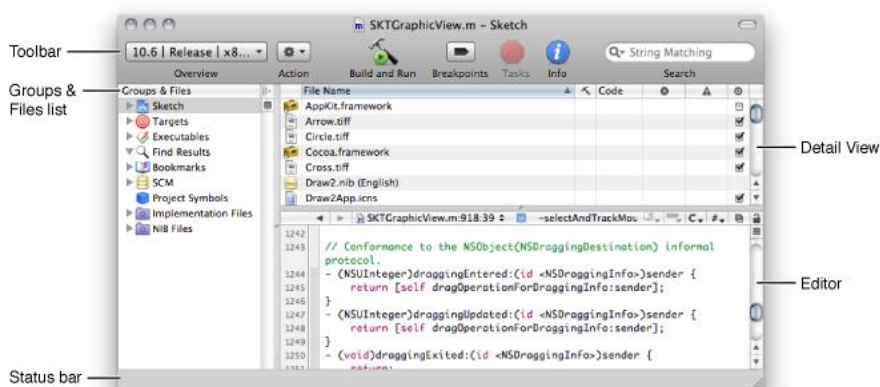


Image 65 – The Xcode Project Window



Objective-C files have either .h, .m or .hh extension. Files with the .h extension are header files which contain class, type, function, and constant declarations. .m files are source files which can contain both Objective-C and C language. .hh files are also source files, but can contain C++ code in addition to Objective-C and C code. Another important file is the .xib file called nib file, which contains the application's user interface. This file is automatically created by the Xcode when a view-based project is created. Nib files cannot be read with a source - or a hex editor, since they are in a proprietary format. Instead, a separate application called Interface Builder is used for designing the UI. (Allan 2010, 31; Apple 2010f.)

Images used in this chapter are from Xcode.

### 3.8.1 Setting up the virtual iPhone device

iOS Simulator comes with the Xcode, so there was really no need to set it up separately. It is used to simulate the behavior of a real device when they are run, and it can simulate the iPhone, iPhone with Retina display and iPad, and several different iOS versions. The 4.3 (238.2) Simulator, which comes with Xcode 3.2.6, has iOS versions 3.2, 4.0.2, 4.1, 4.2 and 4.3 built in. Testing applications on different versions is easy. The debugging console which starts up when a project is built and ran has a menu where all the different versions are found. To run the project on another device or version, the active executable needs to be chosen and the project needs to be built and run again.

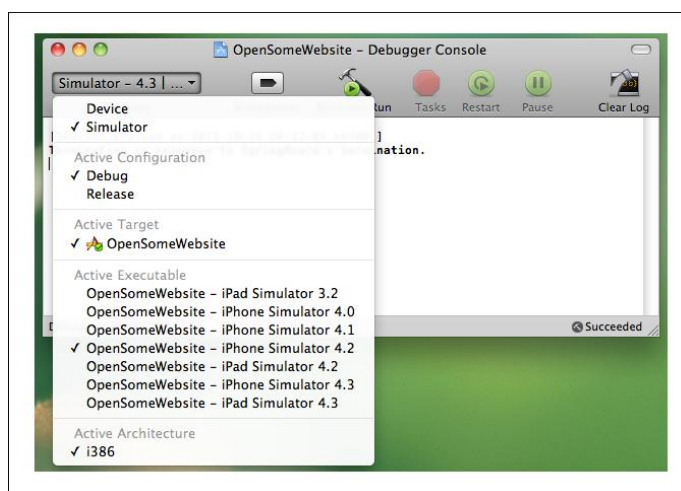


Image 66 – Selecting iPhone Simulator device and version

### 3.8.2 Starting the project in the Xcode environment

When Xcode was run the first time, the welcome screen was displayed. This screen shows the latest projects and makes it easy to open any project that the user has been working with before. Unless the “Show this window when Xcode launches” option is cleared, the same screen is displayed every time Xcode starts up. Of course there is always the option to open a project by double-clicking the project file, and without first opening Xcode. This is another way to prevent the welcome screen from being displayed when the Xcode is started.

The environment was set up for implementing our code after four simple steps:

1. We selected “Create a new Xcode project” from the welcome screen.
2. We chose a template for the project.
3. We chose the target product (i.e. iPhone or iPad).
4. We gave the project a name.



Image 67 – Xcode welcome screen

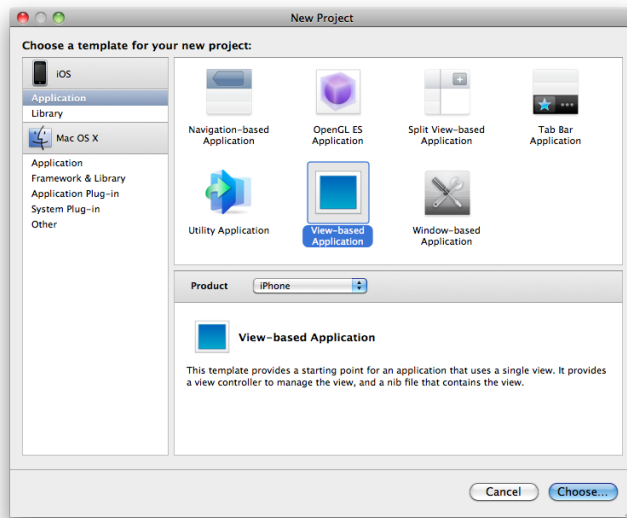


Image 68 – Choosing a template and the target product

Since older iOS versions do not support all the features built in the newer versions, it is a good idea to set the correct Base SDK and iOS Deployment Targets before starting the implementation. Base SDK determines which headers and libraries are used for compiling, and iOS Deployment Target is the minimum iOS version the application is supported on. Configuring these two settings should minimize the risk of implementing code which isn't supported on a device running on older version of the iOS.

We modified both settings by selecting Targets > “*the name of the application*” > Info > Build. Base SDK was selected under the “Architectures” subheading and iOS Deployment Target under the “Deployment” subheading. (Muchov, J. 2010.)

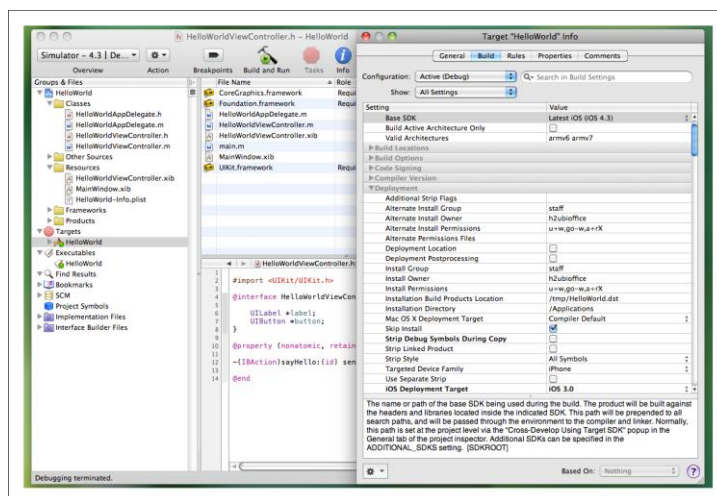


Image 69 – Changing Base SDK and iOS Deployment Target

### 3.8.3 Developing HelloWorld for iOS

Like in the Android part before, the first application we created for iOS was HelloWorld. As everyone with any knowledge of programming knows, HelloWorld is usually the simplest possible kind of software. It is basically used for demonstrating the syntax of a given programming language. Normally its sole purpose is to print out the text “Hello World” and do nothing more.

The behavior of the example demonstrated here is quite similar to that, except that the user first needs to click a button after which the application prints out “Hello, User!” The basic idea is the same, but we just added the button to demonstrate at least some sort of dialog between the user and the application.

As the previous chapter explained, the creation of a new project in Xcode 3.2.6 includes four steps. After creating a new project we began the implementation. Allan’s (2010, 34–38) HelloWorld is the source for our example of the same application. After implementing our example application according to the instructions we ran it. The application was built and run without errors. We used iPhone Simulator with the version 4.3 to see that the application is working the way it should. The source code for the application is found in Appendix B.1. - HelloWorld for iOS.



Image 70 – HelloWorld running on the iPhone simulator

### 3.8.4 Developing OpenSomeWebsite for iOS

The second implementation was an application which uses an internet connection. Like in HelloWorld, OpenSomeWebsite first displays a button, and when the button is clicked the application opens the Safari browser which again opens a website defined in the “OpenSomeWebsiteViewController.h” file, which in this case was chosen as <http://www.telkku.com>.

The application uses the UIApplication class, the sharedApplication class method and the openURL method. UIApplication class is used for controlling and coordinating application running on iOS, and there has to be exactly one instance of it in every application. UIApplicationMain function is called when an application is launched. The function creates a UIApplication object, which can then be accessed by invoking the sharedApplication class method. Because an application can bind itself to a URL, the openURL method can be used to launch various applications, like Google Maps, Apple Mail, and the App Store. However we used it for launching the Safari browser. The source code for the application is found in Appendix B.1. - OpenSomeWebsite for iOS (Aiglstorfer 2008; Apple 2011c.)

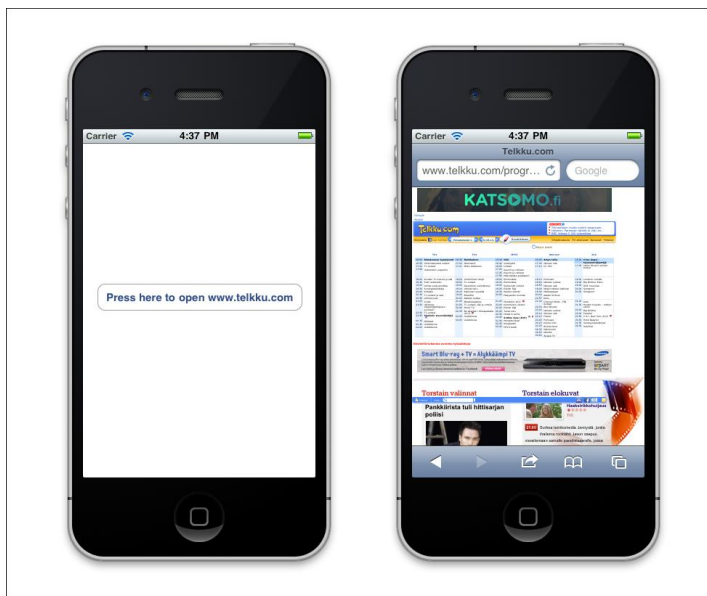


Image 71 – OpenSomeWebsite running on the iPhone simulator (Kustannusosakeyhtiö Italehti 2011.)

### **3.8.5 Summary of the application development process on iOS SDK**

Xcode seems like a simple enough tool for anyone to learn. The outlook of the application is clear and minimalist - as Apple applications often are - and Apple provides thorough documentation for using it. The usability also seems user friendly. In Xcode, there is nothing that really catches the eye so to speak. It feels like everything is where they are expected to be, which of course makes the usability good. Of course learning the syntax of a previously unknown programming language always requires a lot of work, especially for the likes of us who are not that experienced in any kind of programming.

The code completion (sometimes called “intellisense”) works nicely, which makes the daily use of the tool more enjoyable. It’s actually hard to imagine using a develop environment that didn’t have it after one is used to working with the feature. Another very useful feature in for a beginner is the Xcode quick help, which can be opened in the code editor by holding down the “alt option” key and double-clicking a symbol in the editor window. The quick help stays open when the window is moved, and also provides an easy way to access complete reference documentation. The documentation is opened by clicking the book icon in the quick help.

The Interface Builder is consistent with the Xcode design wise. All the tools are easy to find and it is quite to use. The linking of the elements has been made straightforward and understandable.

Unlike the corresponding tool in the Android SDK, not only is the iOS Simulator very fast to start, it also runs fast. As the simulator is very likely used quite often during any kind of development project, it also plays an important role in the development environment.

## **3.9 Deploying an application on iPhone**

Before it was possible to test an application on the iPhone, there were four mandatory steps to be taken. We needed to

1. Designate an iOS device for development and user testing.
2. Create a Certificate Signing Request (CSR).
3. Download and install a digital certificate for development.
4. Create a provisioning profile.

All the devices used in the development need to be added to the iOS Provisioning Portal's Devices –section by the Team Admin. The iOS University program allows 200 devices to be added. If a device is added and later removed it still counts as one and does not reset the device count.

CSR is needed for generating a development certificate, and it is submitted in the iOS Provisioning Portal located in the Member Center of Apple' Developer pages. After the CSR has been approved it is possible to download and install the development certificate. The certificate is associated with the developers name, email address, or business, and is used to identify the developer by using a secret private key and a shared public key. A developer certificate in a keychain includes the private key and the public key is included in the provisioning profile and in the iOS Provisioning Portal. The private key is used by Xcode to sign iOS application binaries, and without the certificate Xcode reports a build error. (Apple 2011d.)

Provisioning Profile links development certificates, devices and iOS application ID's. The provisioning profile uses the development certificate to identify the developer and the device, and it enables the developers of the same team to run applications built by another developer on their devices. Without at least one provisioning profile it's not possible to install iOS applications on a device at all. (Apple 2011d.)

Images used this chapter are from Mac OS X, Xcode, Keychain Access, Apple Developer website and iOS Provisioning Portal.

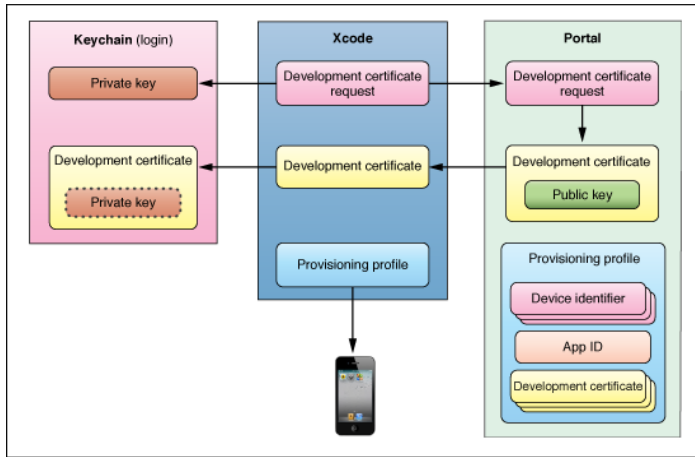


Image 72 – Digital assets required for iOS development

### 3.9.1 Designating iOS devices for development and user testing

Adding iOS devices is done using the Unique Device Identifier (UDID), which is a 40 character string of numbers and letters and is tied to a single device, just like a serial number. We took the identifier from both Xcode’s Organizer utility (Window > Organizer), but it’s also found from iTunes versions 7.7 and later. We set the device for development use from the Organizer’s Summary tab, by clicking “Use for Development”. (Apple 2011d.)



Image 73 – iPhone UDID displayed in Xcode’s Organizer

After finding out the UDID, we added the device to the iOS Provisioning Portal’s Devices collection by selecting “Upload Devices”, entering the device a name and its ID, and clicking the “+” sign next to the Device ID field. There is also the option to add a series of devices by uploading a tab delimited .txt file or using a tool called “iOS Con-



figuration Utility”, which is available in Apple’s iPhone Enterprise Support pages. (Apple 2011d.)

### 3.9.2 Creating a Certificate Signing Request and obtaining the iOS Development Certificate

We began by locating the Keychain Access utility, which in Mac OS X Leopard is found from Applications > Utilities.

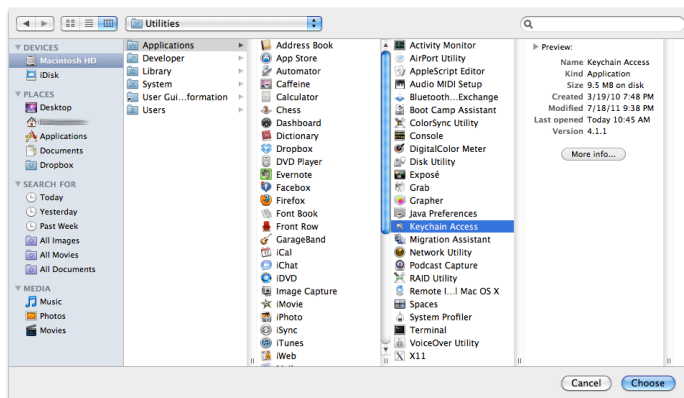


Image 74 – Starting the Keychain Access utility

Following Apple’s How To -instructions found from the iOS Provisioning Portal, we set “Online Certificate Status Protocol (OCSP)” and “Certificate Revocation list (CRL)” to “Off”. These settings are located in the “Preferences” menu in the Keychain Access utility.

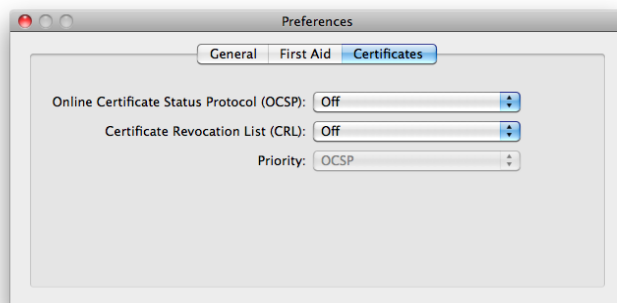


Image 75 – Keychain Access preferences

After the preferences were set, we created the Certificate Signing Request. From Keychain Access we selected the Keychain Access menu > Certificate Assistant > Request a Certificate From a Certificate Authority...

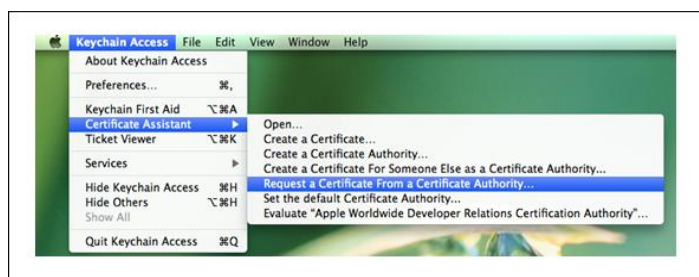


Image 76 – Creating a Certificate Signing Request

Both the User Email Address and Common Name are mandatory fields, and they have to match the information that was used when the iOS Developer registration was submitted. Certificate Authority (CA) Email Address is not required when the request is saved to disk.

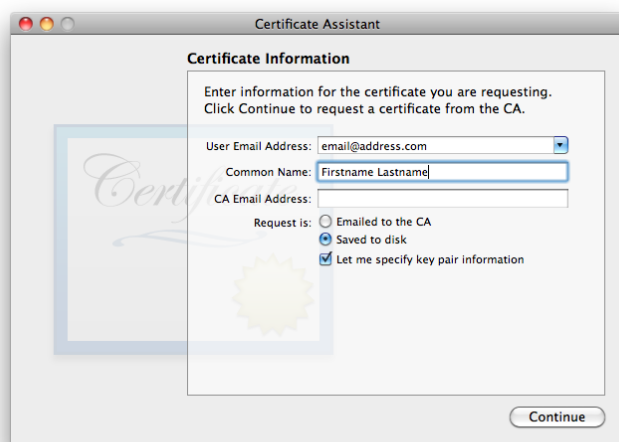


Image 77 – Creating a Certificate Signing Request - certificate information

Since we had selected “Saved to disk” and “Let me specify key pair”, we needed to name the file and choose where the CSR is saved. After selecting “Save”, we were requested to define Key Pair Information. There was no need to change any settings, as by default Key Size was “2048 bits” and Algorithm was “RSA”.

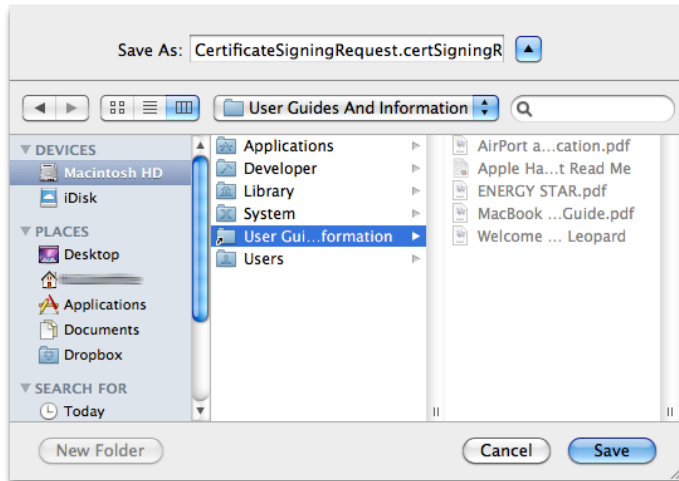


Image 78 – Creating a Certificate Signing Request – saving the CSR

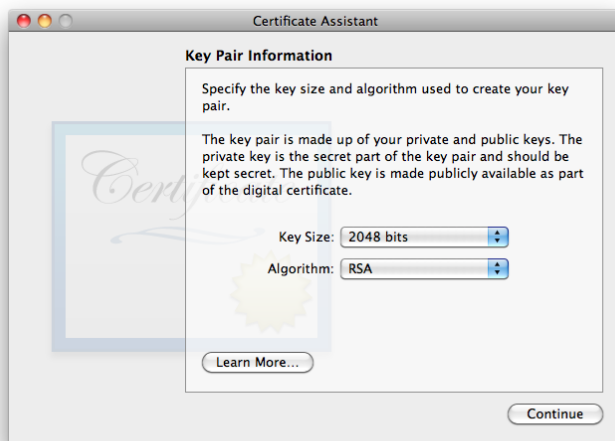
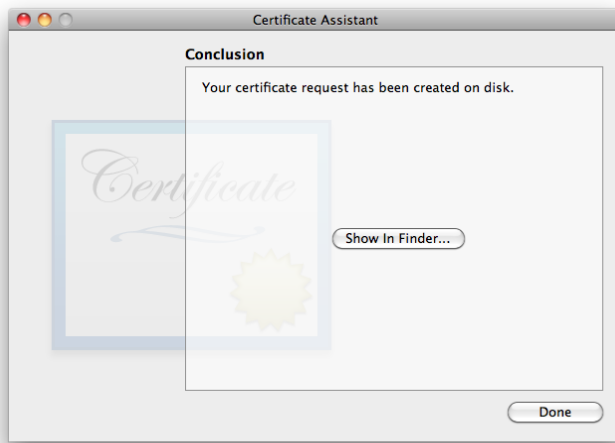


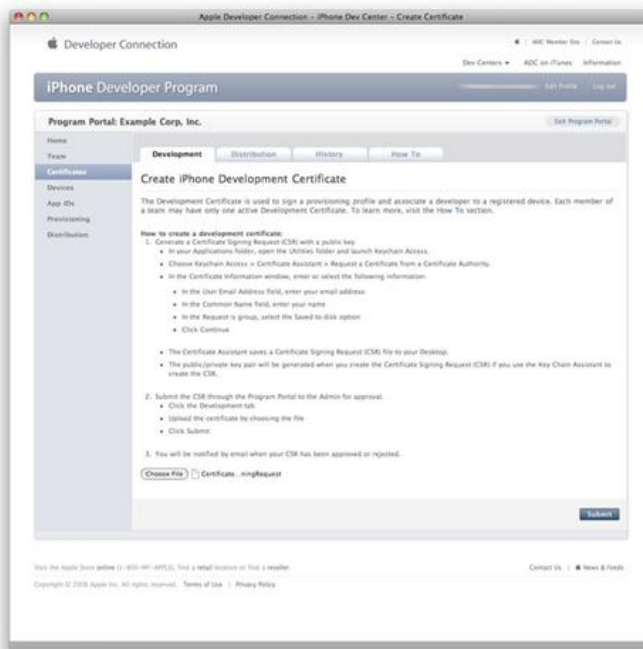
Image 79 – Creating a Certificate Signing Request - defining Key Pair Information

After selecting Continue the request file has been created and a confirmation message is shown.



**Image 80 - Creating a Certificate Signing Request - certificate request created**

After we had created the CSR, we had to submit it in the iOS Provisioning Portal. The portal is found from the Member Center of Apple Developer pages. In the Provisioning Portal we navigated to Certificates > Development, chose the CSR file by selecting “Choose File” and browsing the file from where we saved it, after which we submitted the file by selecting “Submit”



**Image 81 - Submitting the Certificate Signing Request**

Once the request was submitted, our development certificate was shown on the “Development” tab under the “Certificates” menu as a pending certificate. This is where

Team Admins can either reject or approve certificates after they have submitted and approved their own requests. As a Team Agent we approved our development certificate to proceed with the process.

The status of the certificate changed from “Pending Issuance” to “Issued” after the approval and it was available for download.

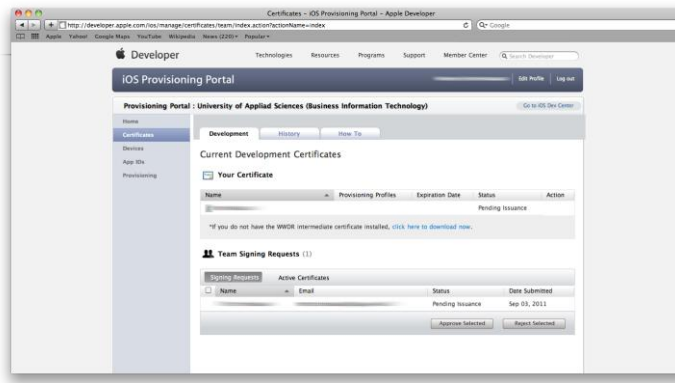


Image 82 – Pending certificates in the iOS Provisioning Portal

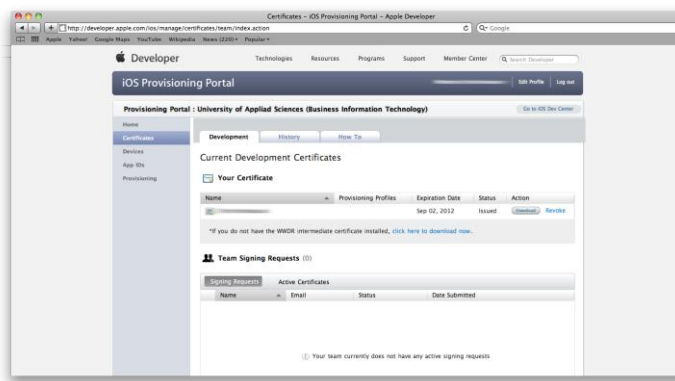


Image 83 – Approved certificates in the iOS Provisioning Portal

Before downloading and installing the developer certificate, we found out that there is also another certificate that needs to be downloaded and stored to the computers Key-chain. It is called “Apple Worldwide Developer Relations Intermediate Certificate” (WWDR) and is used by Xcode to check that the other certificates are valid. This certificate is also found from the iOS Provisioning Profile > Certificates > Development, and can be downloaded from the link: “click here to download now”. Only one WWDR Intermediate is needed per computer.

Of the two certificates we downloaded and installed the WWDR first. Double-clicking the downloaded “AppleWWDRCA.cer” file launched the Keychain Access utility from where the keychain was added to a keychain. After adding, the certificate was shown under “Certificates”.

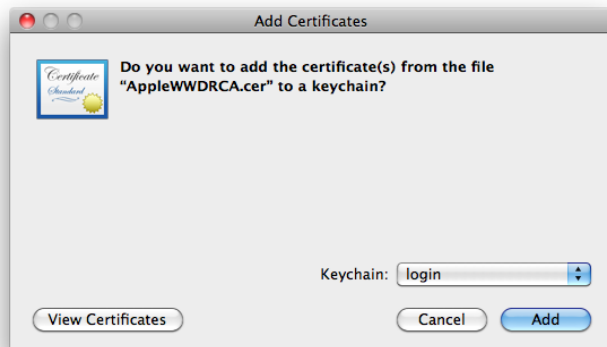


Image 84 – Installing the WWDR Certificate

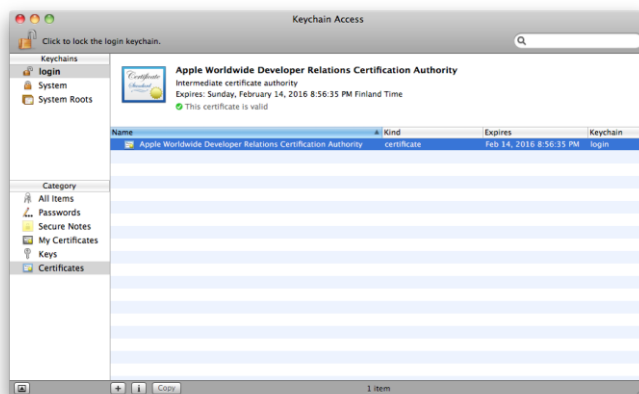
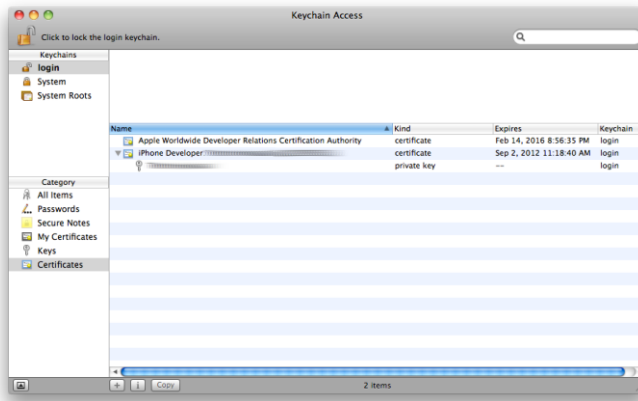


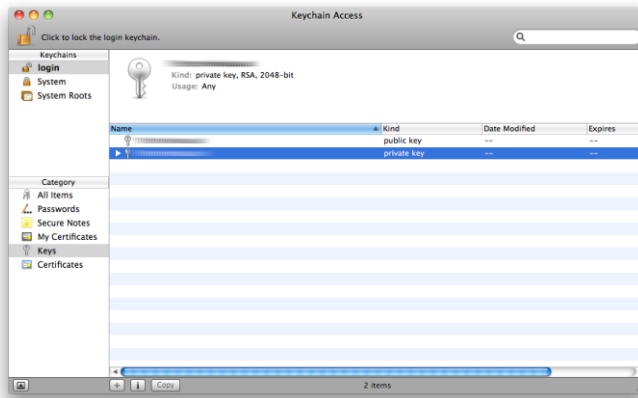
Image 85 – WWDR certificate in Keychain Access

Next, we installed the developer certificate the same way. Double-clicking the “developer\_identity.cer” file installed the certificate. The iPhone Developer certificate was found again from Keychain Access after installation.



**Image 86 – Installed certificates in Keychain Access**

Both keys - public and private key – were then found from “Keys” in Keychain Access, and the process of acquiring both certificates was completed.



**Image 87 – Installed Keys in Keychain Access**

It is important to remember that the private key should be backed up in case of any intention to develop applications on another system or reinstall the operating system later on. The private key cannot be recreated if it is lost at some point and without it, it is impossible to sign binaries in Xcode. We exported the private key by right clicking on the private key in Keychain Access and selecting “Export *name of the certificate*”. Setting a password for the private key was mandatory so we gave it one. The password set here is needed if the private key is imported at some point.

### 3.9.3 Creating and downloading a development provisioning profile

The last step before being able to run an application on the iPhone was to create a provisioning profile from the Provisioning section of iOS Provisioning Portal. We selected “New Profile”, entered a name for the profile, selected the certificate, the App ID and device, and finally submitted the form. The profile was shown immediately on the Provisioning page from where we downloaded it.

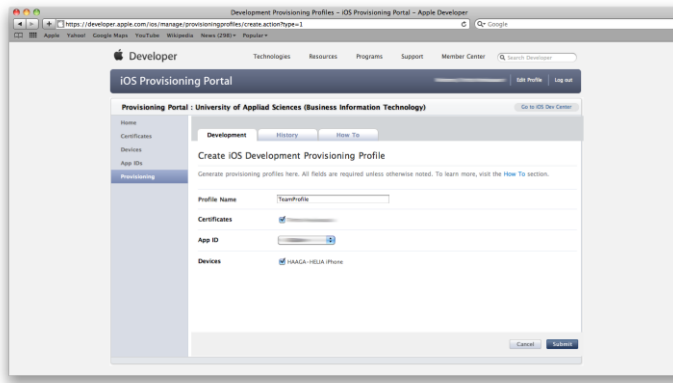


Image 88 – Creating a Provisioning Profile

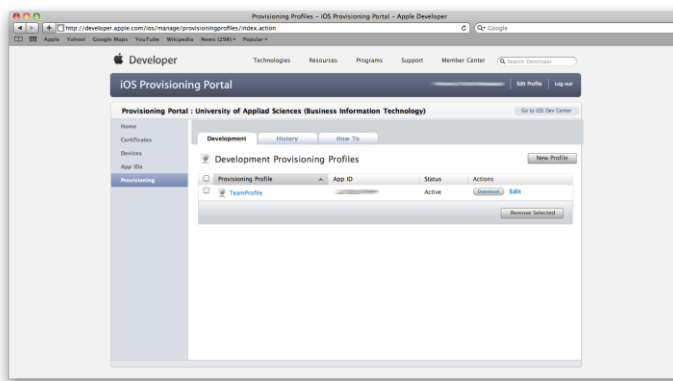


Image 89 – Provisioning Profile created and in Active -status

### 3.9.4 Considering the use of license agreements

All developers whose goal who release their applications on App Store are required to sign the iPhone Developer License Agreement. Even though it is not possible to see this agreement without actually registering to the developer program, Electronic Frontier Foundation (EFF) released a copy of the agreement on 9<sup>th</sup> of March 2010. The



agreement is dated January 2010 so it's not necessarily completely accurate anymore, but is still worth presenting. (Von Lohmann, 2010.)

According to EFF's article the developers themselves have very limited rights to the applications they develop:

Section 7.2 makes it clear that any applications developed using Apple's SDK may only be publicly distributed through the App Store, and that Apple can reject an app for any reason, even if it meets all the formal requirements disclosed by Apple. So if you use the SDK and your app is rejected by Apple, you're prohibited from distributing it through competing app stores like Cydia or Rock Your Phone.

A few other interesting sections of the agreement prohibit the user from reverse engineering the iOS SDK or iPhone OS, and "tinkering with any Apple software or technology." Apple also reserves the rights to "revoke the digital certificate of any of Your Applications at any time", which also allows Apple to remotely disable such apps that have already been installed on users' devices. According to EFF, Section 14 of the agreement states that: "Apple will never be liable to any developer for more than \$50 in damages.", even if an Apple update would for example accidentally break the developer's application. (Von Lohmann, 2010.)

### **3.9.5 Giving the application an icon, a label and a correct version number**

An application is required to have an icon if it is published and added to the App Store. The icon is used to represent the application on the devices home screen. An iPhone application icon should be a 57 x 57 -pixel JPEG or PNG file.

We added the icon to our project by drag & dropping the file under the "Resources" folder, but it can also be added by selecting "Resources" and choosing Project > Add to Project, and then browsing the file from the system. After we had added the file to our project we checked that it also shows in the "Copy Bundle Resources" -field found under Targets > "*name of the application*". Since Xcode doesn't automatically use the picture file as the icon, we also added the filename to the "Icon file" -field, too. This

field is found in Resources > *NameOfTheApplication*-Info.plist > Information Property List. (Apple 2010g.)

Even though a new project has to be named when the development phase begins, it doesn't mean that the end result will carry the same name, nor it should. To make it easy for developers to rename the application later on, Xcode uses variables for certain fields holding information regarding the project. In the same .plist file that was introduced in the previous chapter, there are two fields with the value “\${PRODUCT\_NAME}”: “Bundle display name” and “Bundle name”. The “Product name” - field found under Project > Edit Project Settings > Build > Packaging holds the actual value of “\${PRODUCT\_NAME}”, and when this value is changed, it is automatically changed everywhere in the project where it has been used to represent this value. (Apple 2010h.)

The type configuration and methodology explained above doesn't mean that the display name of an application could not be changed. Quite opposite renaming has been made possible and easy. The use of “Bundle display name” for example enables the user to rename an application making the change only superficial without changing the application's and its folders' names. If a user was to change the name of an application, the “Bundle display name” field's value would change from “\${PRODUCT\_NAME}” to anything the user defined. Since the user only changes the “Bundle display name” - fields value and not the actual “\${PRODUCT NAME}” value, the change does not affect the functionality of the application. (Apple 2010h.)

There are a couple of ways to set and maintain the version numbering of a project. From Xcode the version can be set from the same “Build”-setting as the “Product name” was changed from. Logically the “Current Project Version” -field is found under the “Versioning” -settings. The field allows integer or floating point numbers like 4 or 2.5. “Versioning System” is by default set to “None” and there is no need to change it if this type of versioning is used. (Apple 2010h.)

Another option is to use an Xcode built in versioning tool called “agvtool”. The tool can be used to display, set, and update version information in Xcode projects when Apple generic versioning system is in use. To enable it, “Versioning System” -field has to be changed to “Apple Generic”. After this the project version number can be easily checked or update from Mac OS X’s “Terminal”-utility by browsing to the project folder and using the tool’s commands like “agvtool what-version”, or “agvtool next-version –all”. (Rinaldi 2010.)

### **3.9.6 Archiving and signing the application**

Once the application was ready for sharing we archived it by selecting Build > Build and Archive in Xcode. All archived applications are listed under Organizer’s “Archived Applications” -menu, from where they can be validated, shared and submitted. However, being a member of the iOS University Program only allows sharing the application. It is possible to either distribute the archive for an enterprise, save it to disk or e-mail it. The sharing feature forces the developer to sign the application using the existing certificate when choosing the sharing type.

### **3.9.7 Releasing the application into the App Store**

To be able to release applications into the App Store, the developer needs to be a member of the iOS Developer Program or the iOS Developer Enterprise Program. Since the account used in this research is not part of either of the programs, and the registration fee is \$99/year, it was not possible to release the applications developed for this research. Anyhow, the steps for releasing are explained in the following paragraphs. (Apple 2010e.)

Before an application can be released into the App Store, the team has to have an iOS Distribution Certificate linked to a Distribution Provisioning Profile. Just like with the Development Certificate, a CSR has to be created and approved before the certificate can be downloaded. Only one Distribution Certificate is allowed per team and it must be created and installed by the Team Admin. The Team Admin also has to create a Distribution Provisioning Profile with the following information:

1. Distribution method: App Store.
2. Profile name: [Application\_Name] Distribution Profile.
3. App ID: The appropriate application ID for the application being distributed.

When both the Distribution Certificate and the Distribution Provisioning Profile are in use, the developer has to create an iTunes Connect account. The account uses the same username and password as the iOS Developer Account. iTunes Connect is used to validate the application, so some information about it has to be entered there before releasing. Naturally, the application also has to be signed using the Distribution Certificate. Once the application has passed the validation tests it can be submitted. Again, only Team Admins can release applications into the App Store. (Apple 2011e.)

### **3.10 Legal issues and costs of setting up an iOS SDK environment**

Since iOS is not as open platform as Android, there are some details in the University Program's licenses that we feel are important to know. This chapter introduces some of these license agreements' details. We also calculate some example hardware expenses.

#### **3.10.1 Legal issues**

There are two different Legal Agreements for educational use of iOS SDK: "iOS Developer Program University License Agreement" and "iOS Developer Program University Student Agreement. This chapter combines a few practical points that need to be taken into account when using the iOS SDK for educational purposes.

iOS Developer Program University Agreement includes a University Teaching License which gives a university a permission to teach iOS application development. It also permits sharing of those developed applications with other participants of the course. However it doesn't allow any other kind of distribution, which basically means that they cannot be uploaded to Apple Store without the developer in question being a part of an iOS Developer Program. (Apple 2010i, 1.)

With this agreement the university is permitted to teach courses to “Authorized Students” and install – and permit Authorized Students to install – one copy of the iOS and a “Provisioning Profile” on each “Authorized Test Device”. The number of Authorized Devices depends on how many devices the university has registered and acquired licenses for. The agreement also permits installing a reasonable number of copies of the SDK on Apple computers that are owned or controlled either by the university or “Authorized developers”. Of course the SDK’s are to be used by Authorized Students and Authorized Developers in connection with the course. (Apple 2010i, 4; Apple 2010j, 3)

“Authorized Student” is a student who is enrolled in a course, has an active and valid Registered Apple Developer account, and has agreed to the iOS Developer Program University Student Agreement. (Apple 2010i, 4; Apple 2010j, 3)

“Provisioning Profile” means the provisioning profile provided by Apple for educational purposes and limited distribution of applications for use on Authorized Devices (Apple 2010i, 3).

“Authorized Device” is an iOS device owned or controlled by the university, or owned by an Authorized Developer or an Authorized Student (Apple 2010i, 2).

“Authorized Developer” is university faculty, staff, employee or a contractor who has an active and valid Registered Apple Developer account, and has a need to know or use the Apple Software for teaching purposes (Apple 2010i, 2).

### **3.10.2 Costs of setting up an iOS SDK environment**

Since Apple provides a free license for educational use, the costs of setting up an iOS SDK environment come from hardware expenses. Unfortunately Apple is well known for their policy on pricing their products, and thus such an environment does not necessarily come very cheap. Being unfamiliar with the discount policy of Apple regarding mass orders of tens of devices, it is not possible to take such factors into account when calculating the costs.

At the moment (on 8<sup>th</sup> of August 2011) Apple offers five different kinds of computers; two laptop models, two desktop models and the Mac mini. As Apple Store is usually the cheapest source for any Apple products, we will use their list prices for calculating the prices. Prices in the Apple Store start from the relatively cheap Mac mini (599€–799€) to the really expensive Mac Pro (2.499€–4.999€). (Apple 2011f.)

As stated in a previous chapter, the system and software requirements for running the iOS SDK are a Mac computer with test an Intel processor. This means that all the computers available in the Apple Store are good enough for developing an iOS application. Anyhow, for practical reasons, a desktop computer is probably the smartest choice in a school environment. Apple offers four different iMac desktop options; two with 21.5” displays and two with 27” displays. Their prices range from 1.179€ to 1.929€. The cheaper 21.5” model (1.179€) has a 2.5GHz Quad-Core Intel Core i5 processor, 4GB of memory, 500GB hard drive and an AMD Radeon HD 6750M graphics card with 512MB of memory. The display offers a 1920 x 1080 resolution. 30 21.5” 2.5GHz iMacs would cost a total of 35.370€. (Apple 2011f.)

Another reasonable option price wise is the Mac mini with a non-mac display. The cheaper Mac mini model isn’t as powerful as the iMac, but should run Xcode without problems. It has a 2.3GHz dual-core Intel Core i5 processor, 2GB of memory, a 500GB hard drive and Intel HD Graphics 3000 graphics card, which supports resolution up to 1920 x 1200 (HDMI port) or up to 2560 x 1600 (Thunderbolt port). List price for Mac mini 2.3GHz is 599€, but it doesn’t include a display. The cheapest 21.5” display available in Verkkokauppa.com, <http://www.verkkokauppa.com>, costs 119.90€. It is a 21.5” Acer P226HQVPD with a 1920 x 1080 max resolution, which is the same as with the iMac. Since the Mac mini has a HDMI port and comes with a HDMI to DVI adapter, basically any display device with a HDMI or DVI input can be used with it. 30 2.3GHz Mac minis with 30 Acer P226HQVPD’s would cost a total of 21.567€. (Apple 2011f; Verkkokauppa.com 2011.)

The more expensive Mac mini 2.5GHz has quite similar technical specs with the iMac mentioned; 2.5GHz dual-core Intel Core i5 processor, 4GB of memory, a 500GB hard drive and AMD Radeon HD 6630M graphics card. List price for it is 799€. With the same 21.5” Acer display, 30 2.5GHz Mac minis with 30 Acer P226HQVPD’s would cost a total of 27.567€. (Apple 2011f; Verkkokauppa.com 2011.)

There is also a need for test devices. Not only to test the developed applications, but to also familiarize with how to transfer the application to the device and everything that is needed for that. For this purpose we have calculated the price for 30 test devices; iPhones and iPads.

The cheapest iPhone available in the Apple Store is the older iPhone 3GS (8GB) with a 519€ price tag. The iPhone 4 costs either 629€ (16GB version) or 739€ (32GB version). 30 iPhone 3GS’s would then cost 15.570€ using list prices. 30 iPhone 4’s would cost 18.870€ or 22.170€. (Apple 2011g.)

There are six different models of the iPad2: 16GB, 32GB and 64GB versions of the Wi-Fi only -model and the same size options for the Wi-Fi + 3G version. Wi-Fi models costs 479€ (16GB), 579€ (32GB) or 679€ (64GB). Wi-Fi + 3G versions are a bit more expensive: 599€ (16GB), 699€ (32GB) or 799€ (64GB).

With these list prices, 30 Wi-Fi iPad2’s would cost 14.370€, 17.370€ or 20.370€, and 30 Wi-Fi + 3G iPad2’s would cost 17.970€, 20.970€ or 23.970€. (Apple 2011h.)

Device	Model	Unit price	Price of 30pcs
iMac	21.5" 2.5GHz	1 179,00 €	35 370,00 €
Mac mini + Acer P226HQVPD	2.3GHz + 21.5"	718,90 €	21 567,00 €
Mac mini + Acer P226HQVPD	2.5GHz + 21.5"	918,90 €	27 567,00 €
iPhone 3GS	8GB	519,00 €	15 570,00 €
iPhone 4	16GB	629,00 €	18 870,00 €
iPhone 4	32GB	739,00 €	22 170,00 €
iPad2	16GB Wi-Fi	479,00 €	14 370,00 €
iPad2	32GB Wi-Fi	579,00 €	17 370,00 €
iPad2	64GB Wi-Fi	679,00 €	20 370,00 €
iPad2	16GB Wi-Fi + 3G	599,00 €	17 970,00 €
iPad2	32GB Wi-Fi + 3G	699,00 €	20 970,00 €
iPad2	64GB Wi-Fi + 3G	799,00 €	23 970,00 €

Table 2 – Calculating Apple hardware expenses

## 4 Overall comparison based on the research

In this chapter, we gathered our findings together, performed the comparison between the two operating systems and development environments related, and finally summarized the results. The pros and cons of both operating systems are discussed freely under their own subheading by the person responsible for the given system.

### 4.1 Pros and cons of using Android as a platform

We were both generally very pleased with Android as a mobile operating system. At least from a technically oriented person's point of view, it is very nice that the system is almost fully customizable yet very usable. Despite the technical strengths, Android is not any harder to use in everyday life than any other operating system. The menus are built logically and the application icons clearly demonstrate which application they represent.

Android is an open source system, it has been developed to be like that from the very beginning. This makes it usable with basically every mobile device possible, unless the use of other systems has been prevented in the specific device. The fact that Android is an open source system means that developers and manufacturers can develop the system onwards to suit their own needs. In fact, many manufacturers port the base platform of Android into their user interfaces, an example of this is the Sense GUI made by HTC.

Android is still a growing system in popularity and efficiency and it is being developed constantly. When Android was first released, only the most dedicated mobile device enthusiasts knew about it. Now, during the past few years, Android has gained a bigger share of the market and also the everyday mobile device users have found the system. This is mainly because more manufacturers have started using Android in their devices. As more and more people start using Android, the system is developed onwards more efficiently according to user experiences and at least in our opinion, it is a positive fact that a truly open source system gains a bigger share of the market.



On the other hand, the fact that more manufacturers bring more and more Android devices to the market can result in problems for application developers. At least the most demanding applications, such as 3D games, can be hard to get to function in the cheapest devices: hardware issues will surely arise. The relatively dense rate of updating Android can also be considered a negative factor from the same reason. Even though Android applications are said to be forwards compatible in different platforms, problems can arise especially if an application is developed to be more platform-centered and if new features are built into the new platforms.

In any case, the selection of applications, free or not, available in the Android Market is breath taking. It is possible to find almost anything from the Market. The negative side of this is of course the lack of quality control: anyone can publish their applications. The responsibility falls mainly in the hands of the final user, what is downloaded and what is not.

Regarding the Android SDK and developing programs for Android, we were also quite pleased. The programming language used in developing applications for Android is Java. Java is a very popular programming language, tutorials and instructions can easily be found from literature and Internet articles. The basics of Java are also relatively trivial to learn, the language is quite logical.

The Android SDK has a dedicated team maintaining it. The Android Developers website has everything documented from installing the SDK to developing basic applications. Also many of the different elements of the programming language and Android tools are clearly explained and the use of them documented. It is possible to get very familiar with developing applications for Android only by visiting the Android Developers website.

Unlike iOS developers, the developers publishing their applications for Android Market have quite good control over their products even after releasing. This requires using Android Market Licensing, a feature allowing the publisher to allow or disallow the further use of the product.

Another factor clearly differing from programming for iOS is the cost of developing applications. Unlike iOS, it is basically free of cost to develop applications for Android, the only cost results from the minor developer's fee charged when first registering into the Android Market. Unlike applications for iOS, the applications developed for Android can be distributed from anywhere. It is possible to download the .apk-file from the Internet, for example a personal server share, so it does not have to be released into the Android Market to be downloaded.

## **4.2 Pros and cons of using iOS as a platform**

During the past few years, Apple has built a very strong brand in the mobile market. They have reached a point where other manufacturers always seem to be a couple of steps behind in innovations, even though Apple's products are more or less based on the brand and minimalism. They have gone quite far in creating devices where the control of the device and everything in it - in good and bad - has been taken out of the users hand for the manufacturer to worry about. This is great for those who don't see the need to, for example, modify the operating system, but does not suite those who prefer having as much control over it as possible.

Apple is actually quite known for their policy over their products. Often only Apple products are compatible with other Apple products. This sounds quite restricting, but it also simplifies many things. Since iOS and iOS SDK can only run on Apple's devices, it brings down important everyday issues like system and software requirements. The iOS SDK is also very easy to install because everything that is needed for application development is built into the same installation package, and there is no need for third party applications, add-ons or plugins.

Unfortunately, sometimes it feels like Apple uses their loyal customer base for their own advantage and it might seem unfair for others. They completely own and control the one and only distribution channel for applications, the App Store. Developers may and do create applications, but Apple has the right to say what gets released and what doesn't. They don't need to reason their decisions if they prevent something from be-

ing released and developers just have to accept these decisions. Even if an application gets released, Apple still basically owns the software and can do whatever they want with it.

All the rules also make the application deployment somewhat confusing. Development certificates, distribution certificates, certificates checking other certificates, and different provisioning profiles are just a big mess at first. It takes some time to understand what the purpose of each digital asset is.

To even out all the rules, at least Apple offers a very thorough documentation on their developer site. There are a lot of articles about such general matters as object oriented programming, but also very detailed information like Objective-C syntax guides, example code and information about the iOS architecture. It seems like the whole lifecycle of an application, from beginning application development to releasing it on the App Store, has been quite well documented and guided. But with a comprehensive documentation, it is not very rare to find duplicate information within the documentation, and that sometimes confuses things.

Like in Android's case, iOS applications are forward compatible. If they are developed for an older iOS version they should run on newer versions, too. Of course it is not possible to use API's that the older versions do not support, but with that limitation in mind it is possible to develop software which is compatible with all existing versions. Consistent architecture and user interface between versions is also something the developers should benefit from.

Compared to Android, developing iOS applications is quite expensive with its \$99/year price tag. Of course anyone can become an Apple developer and not join a developer program, but that only allows the access to the actual SDK application and the developer center resources. An Apple developer can't even test applications on a real device nor distribute applications in any way.

The fact that it is not possible to develop applications without a Mac computer is of course very restricting for non-Mac users. Those using a PC have to buy a Mac if they want their part of the Apple hype, and it requires more than just investments in hardware. The user interface, commands and even keyboard shortcuts differ from the PC-, and Windows-world, and getting to know Mac OS X takes its time.

All in all, iOS seems like a solid platform that will be on the market for a long time, and is likely to attract a lot of developers. iOS has a decent market share and an expanding number of users. This even though so far there are only a handful of devices on which the operating system runs: the iPhone, iPad and iPod touch families. If Apple continues to innovate like they have during the last years, their user base is likely to expand and that would bring more customers to 3<sup>rd</sup> party developers. By experience, another positive trend among the Apple users seems to be that they seem to be quite loyal to the brand.

## **5 Discussion about the research**

This chapter consists of our final opinion on how the thesis succeeded as a project and how well we fulfilled our personal learning goals and objectives. During this chapter we considered the success of the methods used, the final research results, the thesis process in general and finally our own learning process. This chapter consists only of our project group members' opinions.

### **5.1 Evaluating the methods**

Our methods for this thesis included theoretical and empirical research. For theoretical research we read related articles, documentation and other literature. The material was mostly provided by the systems' developers, Google and Apple, but also by 3<sup>rd</sup> parties such as authors, and company - and developer bloggers. For empirical research we installed and deployed the SDK environments, developed applications, and ran them on the built in emulator or simulator and finally on the actual mobile devices.

These research methods suited our needs well since our scope consisted of getting familiar with the installation of the SDK's, and development, deployment and releasing process of an application. Being completely new to the mobile development in general, a lot of background work was needed to get familiar with both environments. On the other hand, the vast amount of background work was nicely balanced by the learn-by-doing –type of working method.

Since both of us have a technical background, the idea of first developing and then actually running the applications on real devices was a good motivator for this research. Our background also made it relatively easy to understand the concepts of the processes regarding application development for mobile devices.

### **5.2 Evaluating the research results and their validity**

In our opinion, the research was a success. Throughout the research, we managed to keep to our original objective in everything we did. After conducting the needed back-

ground research, we were able to install both of the SDKs and create and deploy applications with them. We were also successful in finding out the costs and related license issues in using the SDKs in educational environments. In the end, we were able to find a group of similarities and differences between the development environments and we documented all steps taken as was mentioned in the objective.

We feel that we reached the needed results with the research. Our positive feeling was backed up by the commissioners' optimistic comments when the project started nearing its end. We did not fail to gain any of the knowledge mentioned in the objective and we also managed to keep unnecessary factors out of the final research. We feel that the research results are valid until either of the systems are upgraded so far that they are no longer anything like described in this thesis.

### **5.3 Evaluating the overall thesis process**

In general, our whole thesis process was covered without problems. Communication between the steering group and the project group worked well, as did the communication inside the project group. From the beginning of the process to the end we mainly knew what we were doing and if we did not, help was one e-mail message away.

As the scope of the thesis was quite vast, the only way to have finished it in the given timeframe was to conduct the thesis as a pair work. Dividing the work into two equal parts was easy: both of us got our own operating system to work with and which we were responsible for. We had a lot of individual work to do, but also enough pair work, which was no problem in a well-functioning project group.

As a negative side of our thesis process, we were missing our thesis supervisor for most of the project duration. This resulted in us not receiving any guidance regarding the thesis structure until the last few months before the project deadline. We were then appointed with a new thesis supervisor (Juhani Välimäki).

## 5.4 Evaluating the learning process

In general, we learned a lot while conducting this thesis. Neither of us knew the programming languages used with either Android SDK or iOS SDK before this research. In addition to that, we were not at all familiar with developing mobile applications, and we were both completely unfamiliar with the Apple environment. Neither of us had ever even used an Apple product before which created a challenge to begin with.

As both development environments are well documented, it was not difficult to find information. Quite the opposite, sometimes the amount of information was overwhelming, since there was a lot of overlapping within the documentation. We were also forced to be critical and learn how to filter out unwanted information, so we would stay within the scope. In the end, we got very familiar with the developer sites of both systems. As both systems are very popular, it was also easy to find answers to technical questions which rose mainly during the different development phases. Solving these problems was also a good way of learning about the systems.

Towards the end of the project, we felt that we had learned a lot not only about mobile development but also system-specific matters. We felt that we had gained a lot of general knowledge about everything related to the research, ranging from the use of the SDKs to developing and releasing complete mobile applications. We hope the thesis sponsor gains from this research as much as we feel we did.

## **6 Summary and conclusion of the thesis**

In this chapter, we summarize the primary objectives and the purpose of this research and in comparison to them, we briefly review the results. We also discuss our conclusions referring to the primary research problem based on the results obtained.

### **6.1 Summary**

The primary objective of this thesis was to compare Android and Apple iOS software development kits. We were to install the development environments, test them, develop applications with them, publish the applications and finally obtain information on how the development kits could be used in educational environments.

We managed to do all of this, the objective is fulfilled. Both of the development kits have been installed and tested and we also developed two applications with each of the SDKs. This thesis consists of a documentation of the whole process and all the steps taken. We also found distinctive differences between the two SDKs. The differences are also documented in this thesis, mainly in chapter four. From our point of view as a project team, we gathered new theoretical knowledge, and we did our best to document it in this thesis.

### **6.2 Conclusion**

In conclusion, both of the software development kits researched here have their strengths and weaknesses. Both SDKs are equally usable and capable of completing the same tasks, but they are still somewhat out of each other's league. The user of the iOS SDK needs to be an Apple-person at least at some level; it's not possible to develop and test the application without an Apple computer and mobile device. The Android SDK user can enjoy the positive sides of the system being open source; the SDK works on any operating system and the Android system is available for many different devices.



On the other hand, while installing the SDK at first, the Android SDK user needs to do much more work as many different systems need to be combined together, as the iOS SDK basically installs with the click of a button. Though, if taking the application development to the level of actual publishing, Android SDK is easier to use as iOS SDK requires the use of many different digital assets.

As shown, neither of the two SDKs is better than the other on a general level. They both have their good and bad sides, so it is up to the user to decide what is looked for in the system and application to be created and choose the SDK to be used based on that.

## **Bibliography**

### **General sources used**

HAAGA-HELIA University of Applied Sciences 2011a. URL: [http://extra.haaga-helia.fi/english/units/information\\_technology/thesis/Documents/thesis\\_report\\_research.doc](http://extra.haaga-helia.fi/english/units/information_technology/thesis/Documents/thesis_report_research.doc). Quoted: 20.6.2011.

HAAGA-HELIA University of Applied Sciences 2011b. URL: Guidelines for completing a thesis report. URL: [https://extra.haaga-helia.fi/english/studies/guidelines/thesis/Documents/Report\\_Layout\\_and\\_Citing\\_Sources.doc](https://extra.haaga-helia.fi/english/studies/guidelines/thesis/Documents/Report_Layout_and_Citing_Sources.doc). Quoted: 20.6.2011.

Kustannusosakeyhtiö Iltalehti 2011. Telkku.com. URL: <http://www.telkku.com>. Quoted: 29.9.2011.

### **Sources used in the research about Android and Android SDK**

Android Code Monkey 2010. Hello World - Your First Android Application. URL: <http://androidcodemonkey.blogspot.com/2010/01/hello-world-your-first-android.html>. Quoted: 22.9.2011.

Android Developers 2011a. What is Android? URL: <http://developer.android.com/guide/basics/what-is-android.html>. Quoted 15.7.2011.

Android Developers 2011b. Introduction. URL: <http://developer.android.com/guide/developing/index.html>. Quoted 15.7.2011.

Android Developers 2011c. Tools. URL: <http://developer.android.com/guide/developing/tools/index.html>. Quoted 15.7.2011.

Android Developers 2011d. System Requirements. URL:  
<http://developer.android.com/sdk/requirements.html>. Quoted: 1.7.2011.

Android Developers 2011e. Android Developers. URL:  
<http://developer.android.com/index.html>. Quoted: 1.7.2011.

Android Developers 2011f. Download the Android SDK. URL:  
<http://developer.android.com/sdk/index.html>. Quoted: 1.6.2011.

Android Developers 2011g. Installing the SDK. URL:  
<http://developer.android.com/sdk/installing.html>. Quoted: 1.6.2011.

Android Developers 2011h. Hello, World. URL:  
<http://developer.android.com/resources/tutorials/hello-world.html>. Quoted:  
22.9.2011.

Android Developers 2011i. Web View. URL:  
<http://developer.android.com/reference/android/webkit/WebView.html>. Quoted:  
29.9.2011.

Android Developers 2011j. Hello, Web View.  
URL://<http://developer.android.com/resources/tutorials/views/hello-webview.html>.  
Quoted 29.9.2011.

Android Developers 2011k. Preparing to Publish: A Checklist. URL:  
<http://developer.android.com/guide/publishing/preparing.html>. Quoted 25.9.2011.

Android Developers 2011l. UI/Application Exerciser Monkey. URL:  
<http://developer.android.com/guide/developing/tools/monkey.html>. Quoted:  
25.9.2011.

Android Developers 2011m. Application Licensing. URL:  
<http://developer.android.com/guide/publishing/licensing.html>. Quoted: 29.9.2011.

Android Developers 2011n. Signing Your Applications. URL:  
<http://developer.android.com/guide/publishing/app-signing.html>. Quoted: 25.9.2011.

Android Developers 2011o. Publishing on Android Market. URL:  
<http://developer.android.com/guide/publishing/publishing.html#market>. Quoted:  
29.9.2011.

Android Open Source Project 2011. Welcome to Android. URL:  
<http://source.android.com/>. Quoted: 31.7.2011.

DiMarzio, J. 2008. Android: A Programmer's Guide. McGraw-Hill. New York.

The Eclipse Foundation 2011a. Eclipse Downloads.  
<http://www.eclipse.org/downloads>. Quoted: 1.6.2011.

The Eclipse Foundation 2011b. Eclipse Public License. URL:  
<http://www.eclipse.org/org/documents/epl-v10.php>. Quoted: 31.7.2011.

Free Software Foundation 2006. Sun begins releasing Java under the GPL. Press Release. URL: <http://www.fsf.org/news/fsf-welcomes-gpl-java.html>. Quoted: 31.7.2011.

Free Software Foundation 2007. Gnu Operating System. Gnu General Public License. URL: <http://www.gnu.org/copyleft/gpl.html>. Quoted: 31.7.2011.

Google Code 2011. Obtaining a Maps API Key. URL: <http://code.google.com/intl/fi-FI/android/add-ons/google-apis/mapkey.html>. Quoted: 29.9.2011.

Open Handset Alliance 2007a. Frequently Asked Questions. URL:  
[http://www.openhandsetalliance.com/oha\\_faq.html](http://www.openhandsetalliance.com/oha_faq.html). Quoted: 15.7.2011.

Open Handset Alliance 2007b. Members. URL:  
[http://www.openhandsetalliance.com/oha\\_members.html](http://www.openhandsetalliance.com/oha_members.html). Quoted: 15.7.2011.

Oracle 2011a. Java SE Downloads. URL:  
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Quoted  
1.6.2011.

Oracle 2011b. Oracle Binary Code License Agreement for the Java SE Platform Products. <http://www.oracle.com/technetwork/java/javase/terms/license/index.html>.  
Quoted: 31.7.2011.

### **Sources used in the research about Apple iOS and iOS SDK**

Aiglstorfer, R. 2008. Launching Other Apps within an iPhone Application. URL:  
<http://iphonedevopertips.com/cocoa/launching-other-apps-within-an-iphone-application.html>. Quoted 19.10.2011

Allan, A. 2010. Learning iPhone Programming: From Xcode to App Store. O'Reilly Media, Inc. Sebastopol, CA.

Apple 2009. Apple to Ship Mac OS X Snow Leopard on August 28. URL:  
<http://www.apple.com/pr/library/2009/08/24Apple-to-Ship-Mac-OS-X-Snow-Leopard-on-August-28.html>. Quoted: 2.6.2011

Apple 2010-2011. Looking for Xcode 3? Download now Xcode 3 and iOS SDK 4.3 Readme.  
URL: <https://developer.apple.com/xcode/> (Xcode 3 and ios sdk 4.3 readme.pdf).  
Quoted: 27.6.2011

Apple 2010a. iOS Dev Center. iOS Developer Library. Topics. General. iOS Technology Overview. About iOS Development. URL: <https://developer.apple.com/>.  
Quoted: 20.6.2011.

Apple 2010b. iOS Dev Center iOS Developer Library. Topics. Tools & Languages. Introduction. URL: <https://developer.apple.com/>. Quoted: 12.10.2011.

Apple 2010c. Apple 2010. iOS Dev Center. iOS Developer Library. Topics. General. iOS Overview. URL: <http://developer.apple.com/>. Quoted: 16.6.2011.

Apple 2010d. Mac App Store Preview. URL: <http://itunes.apple.com/us/app/xcode/id448457090?mt=12>. Quoted 27.6.2011.

Apple 2010e. Apple Developer Programs. URL: <http://developer.apple.com/programs/>. Quoted 6.6.2011.

Apple 2010f. iOS Dev Center. iOS Developer Library. Topics. Tools & Languages. Learning Objective-C: A Primer. URL: <http://developer.apple.com/>. Quoted: 19.10.2011.

Apple 2010g. iOS Dev Center. iOS Developer Library. Topics. General. iOS App Programming Guide. App-Related Resources. URL: <http://developer.apple.com/>. Quoted: 21.9.2011

Apple 2010h. iOS Dev Center iOS Developer Library Topics. Tools & Languages. iOS App Development Workflow Guide. URL: <http://developer.apple.com/>. Quoted: 11.10.2011

Apple 2010i. iOS Developer Program University License Agreement. Apple Developer Member Center. Organization: University of Applied Sciences (Business Information Technology). Legal Agreements. URL: <http://developer.apple.com/>. Quoted: 28.7.2011

Apple 2010j. iOS Developer Program University Student Agreement

Apple Developer Member Center. Organization: University of Applied Sciences (Business Information Technology). Legal Agreements. URL: <http://developer.apple.com/>. Quoted: 28.7.2011

Apple 2010k. Xcode 3.2.6 and iOS SDK 4.3 installer.

Apple 2011a. URL: <http://developer.apple.com/xcode/>. Quoted: 3.8.2011

Apple 2011b. iOS Dev Center. iOS Developer Library. Topics. General. Tools for iOS Development. URL: <https://developer.apple.com/>. Quoted: 21.9.2011.

Apple 2011c. OS Dev Center. iOS Developer Library. Frameworks. Cocoa Touch Layer. UIApplication Class Reference. URL: <http://developer.apple.com/>. Quoted: 16.10.2011

Apple 2011d. Member Center. iOS Provisioning Portal. URL: <http://developer.apple.com/>. Quoted: XXX

Apple 2011e. iOS Dev Center. iOS Developer Library. Topics. Tools & Languages. iOS App Development Workflow Guide. Distributing Applications. URL: <http://developer.apple.com/>. Quoted: 14.10.2011

(Apple 2011f.) Apple 2011. Apple Store. URL: [http://store.apple.com/fin/browse/home/shop\\_mac/](http://store.apple.com/fin/browse/home/shop_mac/). Quoted: 2.8.2011

Apple 2011g. URL: [http://store.apple.com/fin/browse/home/shop\\_iphone/](http://store.apple.com/fin/browse/home/shop_iphone/). Quoted: 9.8.2011.

Apple 2011h. URL [http://store.apple.com/fin/browse/home/shop\\_ipad/](http://store.apple.com/fin/browse/home/shop_ipad/). Quoted: 10.8.2011.

Bucanek, J. 2010. Professional Xcode 3. Wiley Publishing, Inc. Hoboken, NJ.

Carvell K. 2010. Cisco and Apple Agreement on IOS Trademark. URL:  
[http://blogs.cisco.com/news/cisco\\_and\\_apple\\_agreement\\_on\\_ios\\_trademark/](http://blogs.cisco.com/news/cisco_and_apple_agreement_on_ios_trademark/).  
Quoted: 26.7.2011

Muchov, J. 2010. Developing iPhone Apps with iOS4 SDK, Deploying to 3.x Devices:  
Base SDK and iPhone OS Deployment Target. URL:  
<http://iphonedevloperstips.com/xcode/base-sdk-and-iphone-os-deployment-target-developing-apps-with-the-4-x-sdk-deploying-to-3-x-devices.html>. Quoted: 12.10.2011.

Rinaldi, G. 2010. Setting iOS Application Build Versions. URL:  
<http://gabrielrinaldi.me/blog/2010/10/13/setting-ios-application-build-versions.html>.  
Quoted: 16.10.2011.

Trebitowski, B, Allen, C. & Appelcline S. 2011. iPhone and iPad in Action. Manning  
Publications Co. Stamford, CT.

Verkkokauppa.com 2011. URL:  
<http://www.verkkokauppa.com/fi/product/38692/dbxdm/Acer-P226HQVbd-21-5-Full-HD-LCD-naytto>. Quoted: 8.8.2011

Von Lohmann, F. 2010. All Your Apps Are Belong to Apple: The iPhone Developer  
Program License Agreement. URL: <https://www.eff.org/deeplinks/2010/03/iphone-developer-program-license-agreement-all>. Quoted: 12.10.2011



# Appendix A - Android source code

## Appendix A.1. - HelloWorld for Android

HelloWorldActivity.java:

```
package com.example.helloworld;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button = (Button) findViewById(R.id.button1);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(HelloWorldActivity.this, "Hello, User!",
                Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:layout_width="wrap_content"
        android:text="Please, press here!"
        android:layout_height="wrap_content"
        android:id="@+id/button1"></Button>
</LinearLayout>
```

Strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello User, I am a test application!</string>
    <string name="app_name">HelloWorld</string>
</resources>
```

## Appendix A.2. - OpenSomeWebsite for Android

OpenSomeWebsiteActivity.java:

```
package com.example.opensomewebsite;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class OpenSomeWebsiteActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button = (Button) findViewById(R.id.button1);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Uri uri = Uri.parse("http://www.telkku.com");
                Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                startActivity(intent);
            }
        });
    }
}
```

Main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button android:text="Press here to open www.telkku.com" android:
id:id="@+id/button1" android:layout_width="fill_parent" andro-
id:layout_height="fill_parent"></Button>
</LinearLayout>
```

Strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, OpenSomeWebsiteActivity!</string>
    <string name="app_name">Open Telkku.com</string>
</resources>
```

## Appendix B - iOS source code

### Appendix B.1. - HelloWorld for iOS

#### HelloWorldAppDelegate.h

```
#import <UIKit/UIKit.h>
@class HelloWorldViewController;
@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate>
{
    UIWindow *window;
    HelloWorldViewController *viewController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet HelloWorldViewController
*viewController;
@end
```

#### HelloWorldAppDelegate.m

```
#import "HelloWorldAppDelegate.h"
#import "HelloWorldViewController.h"
@implementation HelloWorldAppDelegate
@synthesize window;
@synthesize viewController;
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
- (void)dealloc
{
    [viewController release];
    [window release];
    [super dealloc];
}
@end
```

#### HelloWorldViewController.h

```
#import <UIKit/UIKit.h>
@interface HelloWorldViewController : UIViewController
{
    UILabel *label;
    UIButton *button;
}
@property (nonatomic, retain) IBOutlet UILabel *label;
- (IBAction)sayHello:(id) sender;
@end
```

## HelloWorldViewController.m

```
#import "HelloWorldViewController.h"
@implementation HelloWorldViewController
@synthesize label;
-(IBAction) sayHello:(id) sender
{
    label.text = @"Hello, User!";
}
@end
```

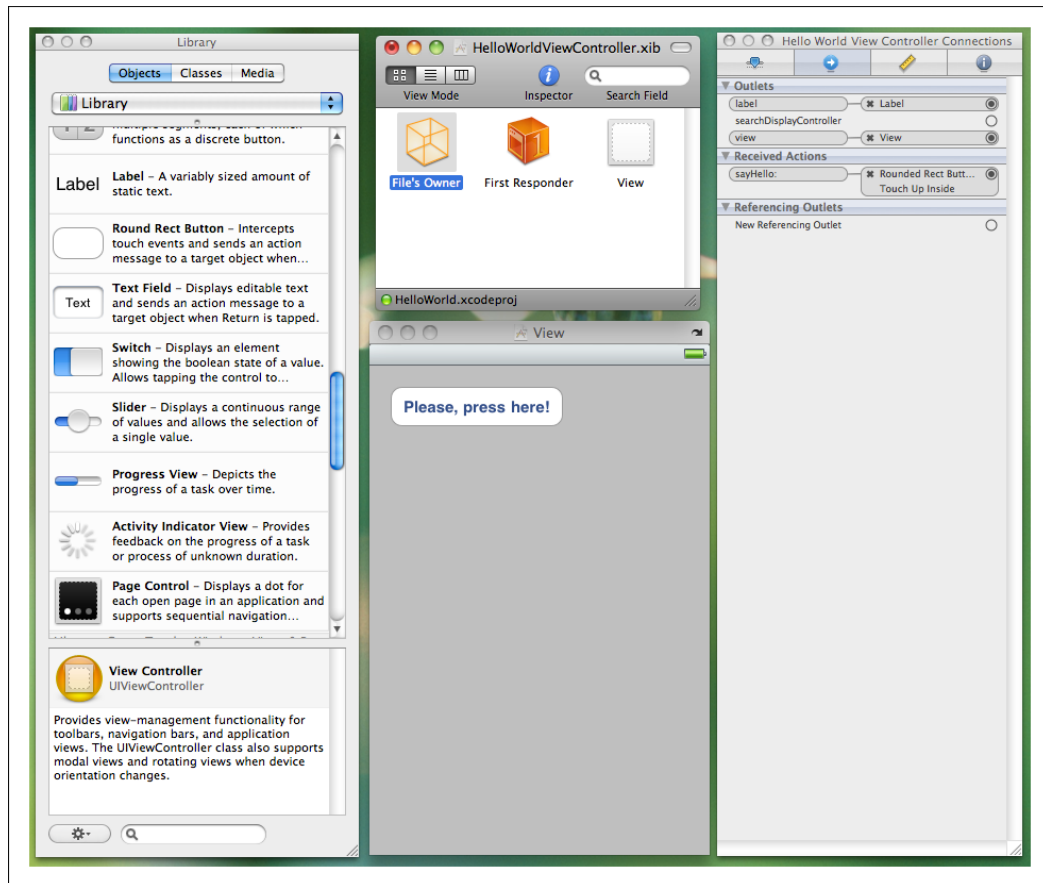


Image 90 – HelloWorldViewController.xib in Interface Builder

## Appendix B.2. - OpenSomeWebsite for iOS

### OpenSomeWebSiteAppDelegate.h

```
#import <UIKit/UIKit.h>
@class OpenSomeWebsiteViewController;
@interface OpenSomeWebsiteAppDelegate : NSObject <UIApplicationDelegate>
{
    UIWindow *window;
    OpenSomeWebsiteViewController *viewController;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) IBOutlet OpenSomeWebsiteViewController *viewController;
@end
```

### OpenSomeWebSiteAppDelegate.m

```
#import "OpenSomeWebsiteAppDelegate.h"
#import "OpenSomeWebsiteViewController.h"
@implementation OpenSomeWebsiteAppDelegate
@synthesize window;
@synthesize viewController;
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window.rootViewController = self.viewController;
    [self.window makeKeyAndVisible];
    return YES;
}
- (void)dealloc
{
    [viewController release];
    [window release];
    [super dealloc];
}
@end
```

### OpenSomeWebsiteViewController.h

```
#import <UIKit/UIKit.h>
@interface OpenSomeWebsiteViewController : UIViewController
{
}
- (IBAction)website;
@end
```

## OpenSomeWebsiteViewController.m

```
#import "OpenSomeWebsiteViewController.h"

@implementation OpenSomeWebsiteViewController

- (IBAction)website

{

    [[UIApplication sharedApplication]openURL:[NSURL URLWithString:@"http://www.telkku.com/"]];

}

@end
```

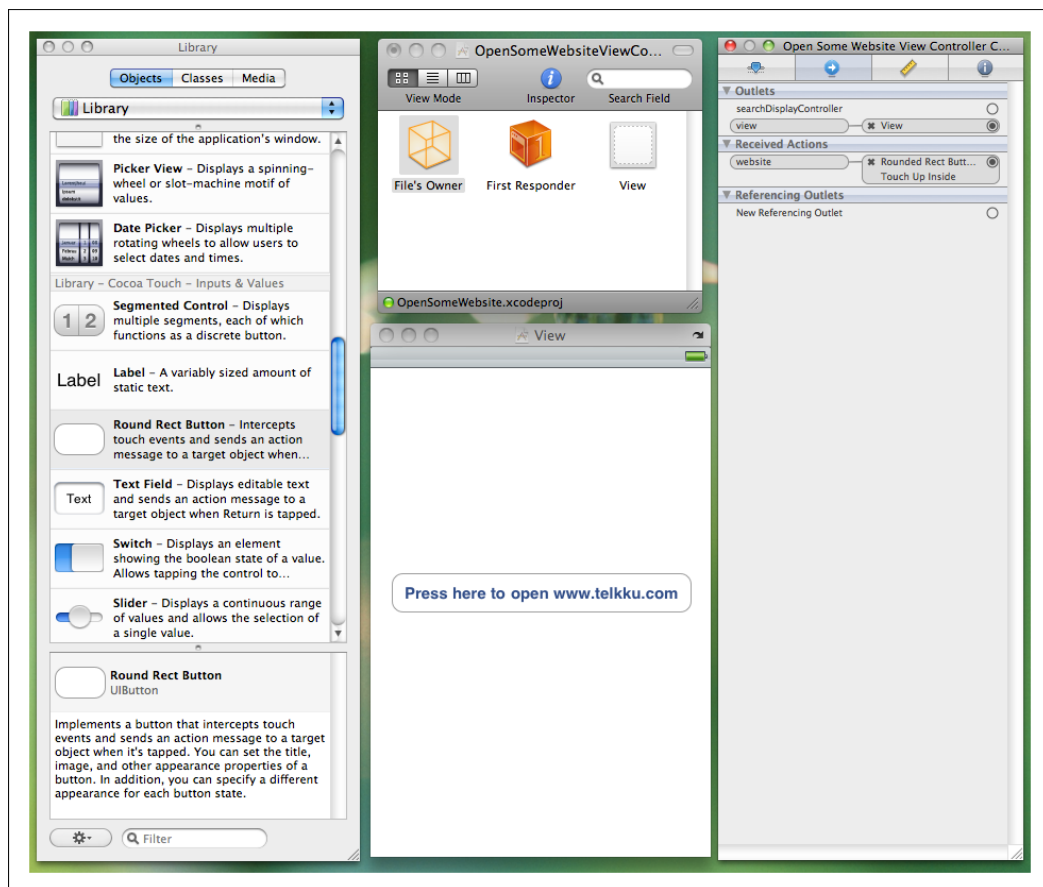


Image 91 – OpenSomeWebsiteViewController.xib in Interface Builder