



Bachelorarbeit 2009

Studiengang Wirtschaftsinformatik

Entwicklung eines Barcodelesers unter Android



Student : Stefan Theler

Dozent : Laurent Sciboz

# Abstract

Diese Bachelorarbeit befasst sich mit der Entwicklung eines Barcodelesers unter dem neuen Handy-Betriebssystem Android.

Sie beschreibt jedoch nicht nur die Applikation selber, sondern auch den Aufbau von Android und seine Architektur. Ebenfalls werden die Komponenten und die Struktur von Android beschrieben und praktisch angewendet.

Der praktische Teil beinhaltet eine Applikation, welche folgende Kriterien erfüllt:

- Mit der integrierten Kamera eines Android fähigen Handys ist es möglich einen 1D oder 2D Barcode (Datamatrix) zu fotografieren und zu analysieren.
- Der Barcode lässt sich in seine Bestandteile zerlegen und gibt die eindeutige Identifikationsnummer des Barcodes zurück
- Durch eine Anbindung an einen externen Server ist es möglich, Produktinformationen zu erhalten und benutzerfreundlich darzustellen.
- Es ist möglich einen Scanvorgang zu speichern, um so später wieder auf dieses Produkt zu zugreifen.
- Die Applikation ist in mehreren Sprachen verfügbar und ist dementsprechend übersetzt.
- Durch das Einfügen eines C-Decoders ist das Auslesen eines Codes sehr zuverlässig und sehr schnell.

Um diese technische Bachelorarbeit zu verstehen, sollte der Leser über folgende Kenntnisse verfügen:

- Kenntnisse in der Java und XML Programmiersprache.
- Grundlegende Kenntnisse im Umgang mit dem Java Native Interface
- Freude an neuen Technologien besitzen.

Dem Leser soll diese Bachelorarbeit folgendes Know-How vermitteln:

- Der Leser versteht Android und seinen Aufbau verstehen
- Der Leser weiss wie die SDK installieren und die Werkzeuge effizient zu nutzen.
- Der Leser ist fähig, eigene Applikationen zu entwickeln.
- Der Leser versteht den Aufbau und die Struktur der CodeCares Applikation.
- Der Leser ist fähig, technische Änderungen an der CodeCares Applikation vorzunehmen.
- Der Leser weiss wie der Android Market aufgebaut ist und wie er diesen verwendet.
- Der Leser ist fähig eine Applikation auf den Android Market zu plazieren und diesen zu veröffentlichen.

## **Eidesstattliche Erklärung**

Ich bestätige hiermit, dass ich die vorliegende Bachelorarbeit alleine und nur mit den angegebenen Hilfsmitteln realisiert habe und dass ich ausschliesslich die erwähnten Quellen benutzt habe. Ohne Einverständnis des Studiengangsleites und des für die Bachelorarbeit verantwortlichen Dozenten sowie des Forschungspartners, mit dem ich zusammengearbeitet habe, werde ich diesen Bericht an niemanden verteilen.

Sierre, den 3. August 2009

.....

Stefan Theler

## **Danksagung**

Danken möchte ich an dieser Stelle Herrn Laurent Sciboz, der mir die Erstellung dieser Diplomarbeit erst möglich gemacht hat. Herr Sciboz war für meine Vorschläge jederzeit offen und gab mir die Chance meine eigenen Ideen umzusetzen und zu entwickeln. Des Weiteren Gebührt grossen Dank Herrn Jean-Jérôme Sarrasin, welcher mir tatkräftig zur Seite stand und mir bei Problemen stets geholfen hat.

Dank auch an Herrn Christophe Abbet, für die Hilfe bei der Entwicklung des C-Decoders.

Zuletzt ein grosses Dankeschön an alle Dozenten und Professoren an der HES-SO Valais, dir mir das notwendige Wissen vermittelt haben.

# Inhaltsverzeichnis

<b>1</b>	<b>EINLEITUNG.....</b>	<b>9</b>
1.1	MOTIVATION.....	9
	<i>Motivation des Autors .....</i>	<i>9</i>
	<i>Motivation des Auftraggebers .....</i>	<i>9</i>
1.2	ZIELE .....	9
1.3	DER AUFTRAGGEBER .....	10
	<i>Allgemeines .....</i>	<i>10</i>
	<i>Die Kompetenzen .....</i>	<i>10</i>
<b>2</b>	<b>ANDROID EINFÜHRUNG.....</b>	<b>11</b>
2.1	WAS IST ANDROID?.....	11
	<i>Android ist ein Software Stack .....</i>	<i>11</i>
	<i>Android Features .....</i>	<i>11</i>
2.2	WELCHE BEREICHE SOLL ANDROID NICHT ABDECKEN? .....	12
	<i>Android ist keine Java ME Implementation .....</i>	<i>12</i>
	<i>Android ist kein Smartphone .....</i>	<i>12</i>
	<i>Android ist nicht Googles Antwort auf das iPhone .....</i>	<i>12</i>
	<i>Android ist nicht nur ein UI wie UIQ3 oder S60 .....</i>	<i>12</i>
2.3	DIE ANDROID TIMELINE.....	13
2.4	DIE INSTALLATION EINER ANDROID APPLIKATION .....	13
	<i>Allgemein .....</i>	<i>13</i>
	<i>Installation mittels Android Market .....</i>	<i>14</i>
	<i>Installation über installAPK .....</i>	<i>14</i>
<b>3</b>	<b>DIE ANDROID ARCHITEKTUR.....</b>	<b>15</b>
3.1	DER ÜBERBLICK .....	15
3.2	DIE SCHICHTEN .....	15
	<i>Der Linux Kernel .....</i>	<i>15</i>
	<i>Die Android Runtime .....</i>	<i>15</i>
	<i>Die C-Libraries .....</i>	<i>16</i>
	<i>Application Framework .....</i>	<i>16</i>
	<i>Applications .....</i>	<i>16</i>
3.3	DAS LAYOUT UND DIE ACTIVITIES .....	17
3.4	DER RESOURCE MANAGER .....	17
	<i>Inhalt von /res/anim/ .....</i>	<i>17</i>
	<i>Inhalt von /res/drawable/ .....</i>	<i>17</i>
	<i>Inhalt von /res/layout/ .....</i>	<i>18</i>
	<i>Inhalt von /res/values/ .....</i>	<i>18</i>
	<i>Die Klasse „R“ .....</i>	<i>18</i>
3.5	DAS LAYOUT PRINZIP / DIE VIEWS .....	20
3.6	DAS MANIFEST -FILE .....	21
	<i>Wozu das Manifest-File? .....</i>	<i>21</i>
	<i>Der Aufbau eines Manifest-Files .....</i>	<i>22</i>
	<i>Die Regeln, die eingehalten werden müssen .....</i>	<i>23</i>
	<i>Intents .....</i>	<i>23</i>
<b>4</b>	<b>DIE ANDROID SDK.....</b>	<b>24</b>

4.1	EINFÜHRUNG.....	24
4.2	VORAUSSETZUNGEN .....	24
4.3	VORBEREITUNGEN .....	24
	<i>Anmerkung</i> .....	24
	<i>Installation von Eclipse</i> .....	25
4.4	SDK-INSTALLATION .....	25
	<i>Download</i> .....	25
	<i>Konfiguration für Eclipse 3.3 (Europa)</i> .....	25
	<i>Konfiguration für Eclipse 3.4 (Ganymede)</i> .....	26
	<i>Konfiguration der Android SDK</i> .....	26
4.5	SDK-TOOLS.....	26
	<i>Der Dalvik Debug Monitor Service (DDMS)</i> .....	26
	<i>Das LogCat</i> .....	27
	<i>Die Traceview Analyse</i> .....	29
	<i>Die Android Debug Bridge</i> .....	29
<b>5</b>	<b>EINFÜHRUNG IN DIE CODEICARE APPLIKATION .....</b>	<b>31</b>
5.1	ALLGEMEINES .....	31
5.2	EINFÜHRUNG IN DIE THEMATIK DER BARCODES .....	31
	<i>Was ist ein Barcode</i> .....	31
	<i>Der Unterschied zwischen Barcode und RFID-Tag</i> .....	31
	<i>Die Unterschiede zwischen Strichode und Datamatrix</i> .....	32
	<i>Warum Barcodeerkennung?</i> .....	32
5.3	EINFÜHRUNGSBEISPIEL .....	33
5.4	DAS PRINZIP DES SCANNENS.....	34
5.5	DIE ARCHITEKTUR DER APPLIKATION.....	35
	<i>Voraussetzungen / Hardware</i> .....	35
	<i>Die Bestandteile der Applikation</i> .....	35
	<i>Die Klassenhierarchie</i> .....	36
	<i>Die Datenbank</i> .....	37
<b>6</b>	<b>TECHNISCHE DETAILS ZUR APPLIKATION.....</b>	<b>38</b>
6.1	MENUHANDLING .....	38
6.2	DAS USERMANAGEMENT .....	39
	<i>Die Registrierung</i> .....	39
	<i>Die Aktivierung</i> .....	39
	<i>Die Verwendung der UserId</i> .....	41
6.3	DAS ERROR-HANDLING .....	42
6.4	DAS BARCODE-HANDLING.....	43
	<i>Allgemein</i> .....	43
	<i>Das Prinzip</i> .....	44
	<i>Die Activity / Das Layout</i> .....	45
	<i>Die Kamera-Funktionalitäten</i> .....	47
	<i>Die Datenverarbeitung</i> .....	49
	<i>Der Response Manager</i> .....	52
	<i>Die Web-View-Anzeige</i> .....	57
6.5	DER NATIVE C-DECODER.....	57
	<i>Was ist JNI</i> .....	57
	<i>Die JNI Problematik</i> .....	58
	<i>Die Installation und Konfiguration der NDK</i> .....	58

	<i>Die Verwendung des Android NDK</i> .....	59
	<i>Das File Android.mk erklärt</i> .....	60
	<i>Der Decoder kurz erklärt</i> .....	61
	<i>Die Shared Library kurz erklärt</i> .....	64
6.6	DAS PREFERENCES HANDLING.....	65
	<i>Allgemein</i> .....	65
	<i>Die Sprachen</i> .....	65
	<i>Die Verwendung der History</i> .....	67
6.7	DAS HISTORY HANDLING.....	69
	<i>Die Idee</i> .....	69
	<i>Die Technik</i> .....	69
	<i>Die Anzeige des Verlaufs</i> .....	70
	<i>Das Erstellen der Datenbank</i> .....	74
	<i>Das Einfügen eines History-Eintrages</i> .....	75
	<i>Das Löschen eines History-Eintrages</i> .....	76
	<i>Das Auslesen eines History-Eintrages</i> .....	76
6.8	DAS VERÖFFENTLICHEN DER APPLIKATION.....	77
	<i>Die Voraussetzungen / Vorbereitungen</i> .....	77
	<i>Das .apk-File erstellen</i> .....	78
	<i>Das digitale Signieren</i> .....	78
<b>7</b>	<b>DIE ZUKUNFTSAUSSICHTEN</b> .....	<b>81</b>
7.1	ALLGEMEINE TECHNISCHE VERBESSERUNGEN.....	81
7.2	UPDATE AUF ANDROID „DONUT“.....	81
7.3	DER VERLAUF / DIE HISTORY.....	81
	<i>Anzeigen der Namen</i> .....	81
	<i>Löschen eines einzelnen Eintrages</i> .....	81
	<i>Nur Tage anzeigen, welche Einträge haben</i> .....	81
7.4	AUSBAU DES SPRACHINTERFACES.....	81
7.5	LOCAL-SEARCH.....	82
7.6	WERBUNG.....	82
<b>8</b>	<b>ANDROID MARKET</b> .....	<b>83</b>
8.1	EINFÜHRUNG.....	83
	<i>Was ist der „Android Market“</i> .....	83
	<i>Android Market Timeline</i> .....	84
8.2	DIE VERÖFFENTLICHUNG VON APPLIKATIONEN.....	84
	<i>Registrierung</i> .....	84
	<i>Veröffentlichung</i> .....	85
	<i>Update</i> .....	87
8.3	KONKURRENZ.....	87
	<i>Apple AppStore</i> .....	87
	<i>Nokia Ovi Store</i> .....	87
8.4	VERGLEICH ANDROID MARKET VS. IPHONE APP STORE.....	88
	<i>Google Trends</i> .....	88
	<i>Medialets-Vergleich nach den ersten 24h</i> .....	89
<b>9</b>	<b>SCHLUSSBERICHT</b> .....	<b>93</b>
9.1	GESTECKTE ZIELE VS. ERREICHTE ZIELE.....	93
9.2	KNOWN BUGS.....	94

---

	<i>Decodierungszeit</i> .....	94
	<i>Funktionsabstürze</i> .....	94
9.3	PERSÖNLICHER KOMMENTAR DES AUTORS.....	94



## Abbildungsverzeichnis

ABBILDUNG 1: MARKET ÜBERSICHTSSEITE .....	14
ABBILDUNG 2: ÜBERBLICK ÜBER DIE ANDROID ARCHITEKTUR.....	15
ABBILDUNG 3: ECLIPSE PLATTFORM .....	24
ABBILDUNG 4: ANZEIGE DES LOGCATS.....	28
ABBILDUNG 5: BEISPIEL EAN UND DATAMATRIX .....	32
ABBILDUNG 6: EINFÜHRUNGSBEISPIEL.....	33
ABBILDUNG 7: DAS PRINZIP DER VERARBEITUNG .....	34
ABBILDUNG 8: ARCHITEKTUR UND KOMPONENTE DER APPLIKATION .....	35
ABBILDUNG 9: KLASSENHIERARCHIE .....	36
ABBILDUNG 10: HAUPTMENÜ DER APPLIKATION.....	38
ABBILDUNG 11: FEHLERMELDUNG .....	42
ABBILDUNG 12: DAS PRINZIP DES SCANNENS .....	44
ABBILDUNG 13: VERANSCHAULICHUNG KAMERA-OBERFLÄCHE .....	46
ABBILDUNG 14: KLASSEN-AUFTEILUNG DES VERLAUFS.....	69
ABBILDUNG 15: VERLAUFSANZEIGE .....	70
ABBILDUNG 16: EXPORTIEREN EINER APPLIKATION.....	78
ABBILDUNG 17: ANDROID MARKET.....	83
ABBILDUNG 18: SUCHVOLUMEN INDEX APP STORE UND ANDROID MARKET..	89
ABBILDUNG 19: DOWNLOADSTATISTIK DER ERSTEN 24 .....	90
ABBILDUNG 20: APPSTORE VS. MARKET - KATEGORIENVERGLEICH.....	91

## Tabellenverzeichnis

TABELLE 1: ANDROID TIMELINE .....	13
TABELLE 2: INHALT VON /RES/VALUES .....	18
TABELLE 3: THREAD ZUSTÄNDE .....	27
TABELLE 4: POSTVARIABLEN .....	55
TABELLE 5: RESULTATMÖGLICHKEITEN DES SERVERS ALS STRING .....	56
TABELLE 6: ORDNERSTRUKTUR NDK .....	59
TABELLE 7: INHALT /SOURCES/CODEICARE/ .....	60
TABELLE 8: DATENBANKTABELLE CODEICARE_HISTORY .....	74
TABELLE 9: PARAMETER ZUM SIGNIEREN .....	79
TABELLE 10: ZERTIFIKATSDetails .....	79
TABELLE 11: ANDROID MARKET TIMELINE .....	84
TABELLE 12: MARKET REGISTRATION - UPLOAD ASSETS .....	85
TABELLE 13: MARKET REGISTRATION - LISTING DETAILS .....	85
TABELLE 14: MARKET REGISTRATION - PUBLISHING OPTIONS .....	86
TABELLE 15: MARKET REGISTRATION - CONTACT INFORMATION .....	86
TABELLE 16: MEDIALETS KATEGORIENVERWALTUNG .....	91

## Codeverzeichnis

CODE-ABSCHNITT 3-1: BEISPIEL EINER ANDROID R. KLASSE .....	19
CODE-ABSCHNITT 3-2: BEISPIEL EINES LINEARLAYOUTS .....	20
CODE-ABSCHNITT 3-3: BEISPIEL EINES MANIFEST-FILES .....	22
CODE-ABSCHNITT 4-1: BEISPIEL EINES TRACEVIEW TRACKINGS .....	29
CODE-ABSCHNITT 6-1: ERSTELLEN EINES MENÜS .....	38
CODE-ABSCHNITT 6-2: ERZEUGUNG EINER USERID IN USER.JAVA .....	40
CODE-ABSCHNITT 6-3: AUSLESEN EINER USERID .....	41
CODE-ABSCHNITT 6-4: ERZEUGEN EINES TOAST-ERRORS .....	42
CODE-ABSCHNITT 6-5: ERZEUGEN EINES ALERTDIALOG-ERRORS .....	43
CODE-ABSCHNITT 6-6: ZUM FULLSCREEN MODUS WECHSELN .....	45
CODE-ABSCHNITT 6-7: ERSTELLEN DER RECHTECKE .....	46
CODE-ABSCHNITT 6-8: BEISPIEL EINES PAINT OBJEKTES .....	47
CODE-ABSCHNITT 6-9: AUTOFOCUS-CALLBACK FUNKTION .....	48
CODE-ABSCHNITT 6-10: PREVIEWFRAME-CALLBACK FUNKTION .....	49
CODE-ABSCHNITT 6-11: YUV420SP – BYTE ARRAY IN EIN BITMAP WANDELN ..	50
CODE-ABSCHNITT 6-12: VOR BEREITEN DES DECODIERENS .....	51
CODE-ABSCHNITT 6-13: CODE AN KAMERA ACTIVITY SENDEN .....	52
CODE-ABSCHNITT 6-14: HANDLER ZUM VERARBEITEN DES RESULTATES .....	53
CODE-ABSCHNITT 6-15: AUFRUF DES RESPONSEMANAGERS .....	54
CODE-ABSCHNITT 6-16: GENERIERUNG DER POST-VARIABLEN .....	55
CODE-ABSCHNITT 6-17: POST REQUEST DURCHFÜHREN .....	55
CODE-ABSCHNITT 6-18: ÖFFNEN DES PRODUKTES .....	57
CODE-ABSCHNITT 6-19 ERSTELLEN DER NDK-LIBRARIES .....	60
CODE-ABSCHNITT 6-20: C-DECODIERUNG EINES BARCODES (TEIL 1) .....	61
CODE-ABSCHNITT 6-21: C-DECODIERUNG EINES BARCODES (TEIL 2) .....	62
CODE-ABSCHNITT 6-22: C-DECODIERUNG EINES BARCODES (TEIL 3) .....	62
CODE-ABSCHNITT 6-23: C-DECODIERUNG EINES BARCODES (TEIL 4) .....	63
CODE-ABSCHNITT 6-24: C-DECODIERUNG EINES BARCODES (TEIL 5) .....	63
CODE-ABSCHNITT 6-25: NATIVE DECODIER-METHODE .....	64
CODE-ABSCHNITT 6-26: ERSTELLUNG DER SPRACH-PRÄFERENZ .....	66
CODE-ABSCHNITT 6-27: EINBINDUNG DER SPRACHWAHL .....	67
CODE-ABSCHNITT 6-28: EINBINDUNG DER HISTORY PREFERENCE .....	68
CODE-ABSCHNITT 6-29: ERSTELLEN DER HISTORY (TEIL1) .....	70
CODE-ABSCHNITT 6-30: ERSTELLEN DER HISTORY (TEIL2) .....	71
CODE-ABSCHNITT 6-31: ERSTELLEN DER HISTORY (TEIL3) .....	71
CODE-ABSCHNITT 6-32: ERSTELLEN DER HISTORY (TEIL4) .....	72
CODE-ABSCHNITT 6-33: ERSTELLEN DER HISTORY (TEIL5) .....	73
CODE-ABSCHNITT 6-34: ERSTELLEN DER HISTORY (TEIL6) .....	74
CODE-ABSCHNITT 6-35: SQL-STATEMENT ZUM ERSTELLEN DER TABELLE .....	75
CODE-ABSCHNITT 6-36: EINFÜGEN EINES HISTORY-EINTRAGES .....	75
CODE-ABSCHNITT 6-37: SQL-STATEMENT ZUM ERSTELLEND ER TABELLE .....	76

## Abkürzungsverzeichnis

<b>3G</b>	3rd Generation
<b>API</b>	Applicatoin programming interface
<b>DDMS</b>	Dalvik Debug Monitor Service
<b>DLL</b>	Dynamic Link Library
<b>DVM</b>	Dalvik Virtual Machine
<b>EDGE</b>	Enhanced Data rates for GSM Evolution
<b>GEPiR</b>	Global Electronic Party Information Register
<b>GPRS</b>	General Packet Radio Services
<b>GPS</b>	Global Positioning System
<b>GUI</b>	Graphical User Interface
<b>IDE</b>	Integrated Development Environment
<b>JAR</b>	Java Archive Compressed file archive
<b>Java EE</b>	Java Enterprise Edition
<b>Java ME</b>	Java Mobile Edition
<b>JDK</b>	Java Development Kit
<b>JNI</b>	Java Native Interface
<b>JVM</b>	Java Virtual Machine
<b>PK</b>	Primary Key
<b>RFID</b>	Radio Frequency Identification
<b>SDK</b>	Software Development Kit
<b>SMS</b>	Short Message Service
<b>SQL</b>	Structured Query Language
<b>SSL</b>	Secure Sockets Layer
<b>UI</b>	User Interface
<b>UIQ3</b>	User Interface Quartz 3
<b>URL</b>	Uniform Resource Locator
<b>USB</b>	Universal Serial Bus
<b>VM</b>	Virtual Machine
<b>XML</b>	Extensible Markup Language

# 1 Einleitung

## 1.1 Motivation

### Motivation des Autors

Die Motivation für den Autor bestand hauptsächlich darin, dass diese Bachelorarbeit in die Bereiche „neue Technologien“ eingegliedert ist. Handyapplikationen, welche mit einem Server agieren und dem Benutzer Real-Time Informationen anzeigen wachsen stetig. Zudem ist Android ein Handy-Betriebssystem, welchem die Zukunft gehört. Da der Autor bereits vor dieser Arbeit die Absicht hatte Android zu verstehen und zu lernen, war die Möglichkeit, dies im Rahmen der Bachelorarbeit zu tun, natürlich sehr interessant. Auch Kenntnisse in der Entwicklung von verteilten Systemen können für das spätere Berufsleben nur von Vorteil sein. Des Weiteren hat der Autor bereits im Rahmen einer Semesterarbeit eng mit dem RFID-Center (Institut Icare) zusammengearbeitet und konnte so bereits einige Erfahrungen im Umgang mit Handyapplikationen machen. So war die Wahl schlussendlich eindeutig.

### Motivation des Auftraggebers

Mit mehr als 3 Milliarden Handynutzern weltweit, ist das Mobiltelefon zu einem wichtigen Bestandteil in der Kommunikation und Integration zwischen Unternehmen und Verbrauchern geworden. Heutzutage können Handys Barcodes lesen und sie können sich auch mit dem Internet verbinden. CodeIcare nutzt diese beiden Technologien um Produktinformationen benutzerfreundlich zu präsentieren.

## 1.2 Ziele

Das Ziel dieser Arbeit ist schlussendlich die Entwicklung eines Barocodesers unter Android. Doch bis dieses Ziel erreicht werden kann, muss zuerst die ganze Android Plattform studiert und erlernt werden, sowie die verschiedenen Syntaxformen und Layout-Guidelines.

Am Schluss der Bachelorarbeit soll ein vollständiger Bericht über die Android Plattform, sowie eine technische Dokumentation über die Applikation erstellt werden.

## 1.3 Der Auftraggeber

### Allgemeines

Das Institut Icare ist ein Kompetenzzentrum, das sich auf das „Internet der Dinge“ und auf Mobile-Services spezialisiert hat. Die Entwicklung von neuen Technologien stehen im Vordergrund. Das auf RFID-Technologie spezialisierte, RFID-Center wurde beispielsweise aus dem Institut Icare gegründet.

Das Unternehmen arbeitet seit vielen Jahren eng mit nationalen und internationalen Universitäten und Hochschulen zusammen.

### Die Kompetenzen

Die Aufgabengebiete des Icare Institutes sind in folgende Bereiche gegliedert:

- Analyse, Design und Entwicklung von Informationssystemen für Unternehmen und Öffentliche Verwaltungen.
- Entwicklung von Verfahren zur Bewertung der vorhandenen Informationssysteme.
- Design und Implementierung von Mensch/Maschinen-Schnittstellen
- Ausbau der IT-Sicherheit in Unternehmen

Ein wichtiges Produkt des Instituts stellt die Applikation „codeIcare“ dar, die schon in mehreren Versionen erhältlich ist. Eine J2ME-, sowie eine Version für das iPhone sind bereits verfügbar. Da eine Android-Version bislang fehlt, hat sich das Institut an die Hochschule Wallis gewendet, um in Zusammenarbeit diese Applikation zu entwickeln.

## 2 Android Einführung

### 2.1 Was ist Android?

#### Android ist ein Software Stack

Android besteht aus einem Betriebssystem, aus Middleware und aus Anwendungen. Kurz gesagt: In der Android-Umgebung lassen sich Anwendungen für Handys ausführen.

Das Betriebssystem wurde für Mobiltelefone entwickelt. Seit dem 15. Mai 2009 ist Android auch offiziell für Netbooks<sup>1</sup> erhältlich. Live-Android lässt sich ab CD starten und ist für alle x86 Rechner verfügbar. Dies dient vor allem zum Austesten der Funktionalitäten von Android. Zurzeit kann die Version 0.2 heruntergeladen werden.

Bereits werden einige Netbooks mit Android als installiertem Betriebssystem vertrieben.

#### Android Features

Einige Android Features kurz aufgezählt:

Android...

- ist ein Application Framework
- arbeitet mit der Dalvik Virtual Machine
- besitzt einen integrierter Browser der auf WebKit basiert
- unterstützt Mobiltelefon-optimierte Grafiken
- verwendet SQLite für die Sicherung von Daten
- unterstützt Kamera-, GPS- und Kompass-Manipulationen
- verbindet sich mit Bluetooth, EDGE, 3G und WiFi
- besitzt eine vollumfängliche Entwicklungsumgebung inklusive Emulator und ein Eclipse-Plugin

---

<sup>1</sup> <http://code.google.com/p/live-android/>

## 2.2 Welche Bereiche soll Android nicht abdecken?

### Android ist keine Java ME Implementation

Zwar werden Android Applikationen voll und ganz in Java programmiert, doch sie können nicht unter der Java ME VM ausgeführt werden. So können auch „normale“ Java Klassen und beispielsweise JAR Files nicht auf einem Android Smartphone gestartet werden.

### Android ist kein Smartphone

Anders als das iPhone ist Android kein eigentliches Smartphone. Android ist lediglich eine Plattform, die Betriebssystem, User Interface und Applikationen verbindet. Es gibt kein eigentliches „AndroidPhone“, denn das Betriebssystem kann man auf verschiedenen Smartphones portieren.

### Android ist nicht Googles Antwort auf das iPhone

Das iPhone wurde und wird von nur einer einzigen Unternehmung entwickelt und vertrieben – nämlich Apple.

Der Software Stack Android wurde von der Open Handset Alliance entwickelt. Google steht dieser Allianz lediglich vor. Viele Unternehmen, wie zum Beispiel Vodafone, T-Mobile, Sony Ericsson und LG gehören dieser Allianz an. Unter [http://www.openhandsetalliance.com/oha\\_members.html](http://www.openhandsetalliance.com/oha_members.html) kann die vollständige und aktuelle Liste der Open Handset Alliance betrachtet werden.

### Android ist nicht nur ein UI wie UIQ3 oder S60

Anders als UIQ3 oder S60 ist Android nicht nur eine Benutzeroberfläche für Smartphones. UIQ3 oder S60 bauen auf ein Stack auf und stellen lediglich die Benutzeroberfläche dar. Android hingegen bildet eine Plattform, welche ein Betriebssystem, ein User Interface und verschiedene Applikationen beinhaltet.



## 2.3 Die Android Timeline<sup>1</sup>

**Tabelle 1: Android Timeline**

05.11.2007	Open Handset Alliance kündigt das neue Betriebssystem Android an.
12.11.2007	Entwickler können den ersten Blick auf die SDK werfen und ihre Meinungen mittels Feedback abgeben.
28.08.2008	Die 20 Gewinner der Android Developer Challenge werden bekanntgegeben und der Android Market wird angekündigt.
23.09.2008	Das erste Android-fähige Mobiltelefon, das T-Mobile G1 wird angekündigt und die Version 1.0 der SDK wird zum Download angeboten.
22.10.2008	T-Mobile G1 wird in den USA eingeführt.
02.02.2009	Die Android SDK 1.1 wird zum Download freigegeben
27.04.2009	Die Android SDK 1.5 wird zum Download freigegeben
25.06.2009	Das Android Native Development Kit (NDK) 1.0 wird freigegeben

## 2.4 Die Installation einer Android Applikation

### Allgemein

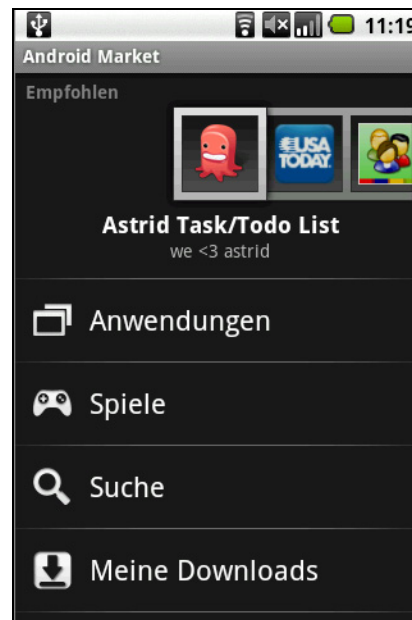
Applikationen lassen sich normalerweise nur mittels des mitgelieferten Market-Tools auf das Smartphone herunterladen und installieren. Dementsprechend ist es eigentlich nicht erlaubt, eine gewünschte Applikationen vom Rechner aus herunterzuladen und später mittels USB auf dem Smartphone zu installieren. Mittlerweile existieren einige Freeware Applikationen, welche auch die Installation mittels externer Programme ermöglichen.

---

<sup>1</sup> <http://www.android.com/timeline.html>

## Installation mittels Android Market

Abbildung 1: Market Übersichtsseite



Quelle: Screenshot HTC Magic

Auf jedem Android-fähigen Smartphone ist die Android Market Applikation bereits vorinstalliert. Sie befindet sich entweder auf dem Desktop oder im Programm-Menü.

Nach dem Öffnen der Applikation erscheint ein Menü mit den Bereichen „Anwendungen“, „Spiele“, „Suche“ und „Meine Downloads“. Im oberen Teil des Bildschirms erscheint eine Liste mit verschiedenen Empfehlungen von Google.

Wenn man nun einen Bereich (Anwendungen oder Spiele) wählt, erscheinen die entsprechenden Kategorien. Klickt man auf die gewünschte Kategorie kann der Benutzer nun die entsprechende Applikation herunterladen. Die Liste kann nach Beliebtheit, sowie nach

Datum sortiert werden.

Falls der Benutzer eine Applikation auswählt, werden ihm verschiedene Informationen und Kommentare (falls vorhanden) angezeigt. Nach einem Klick auf „Installieren“ ist sofort ersichtlich, auf welche Dienste die Applikation zugreifen wird. Nach der Bestätigung dieser Meldung wird das Programm heruntergeladen und installiert. Der Benutzer wird benachrichtigt, wenn die Installation erfolgreich abgeschlossen wurde.

## Installation über installAPK

Die Applikation *installAPK* ist für Windows Rechner entwickelt worden und lässt sich über folgende Webseite herunterladen:

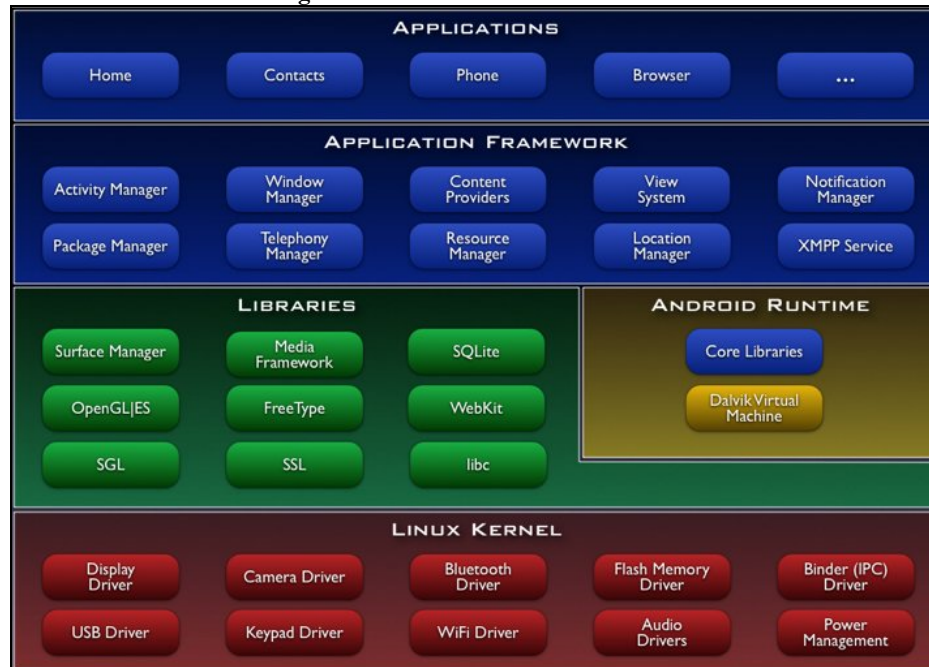
<http://android.modaco.com/content/software/288682/install-apks-on-your-device-by-double-clicking-on-your-pc-with-installapk-beta-1/> (Englisch, letzter Abruf: 23. Juli 2009)

Die Installation startet nach einem Doppel-Klick auf das Installer-File automatisch und informiert den Nutzer nach deren Beendigung. Nun muss das .apk-File mit dem Programm *installAPK* gestartet werden. Dieser Schritt wird mit einem Rechts-Klick auf das File und der Auswahl von „Open with...“ getätigt. Anschliessend wird das Programm automatisch installiert.

## 3 Die Android Architektur<sup>1</sup>

### 3.1 Der Überblick

Abbildung 2: Überblick über die Android Architektur



Quelle: [www.alisdn.com](http://www.alisdn.com)

Android ist ein Software-Stack, welcher alle wichtigen Handy-Komponenten vereint. Die Architektur des Android Betriebssystems ist in 5 Schichten aufgeteilt. In die einzelnen Schichten bauen auf den Linux Kernel 2.6 auf.

### 3.2 Die Schichten

#### Der Linux Kernel

Der Kernel bildet die Hardware-Schicht des Betriebssystems und stellt so die Schnittstelle zwischen Software und Hardware dar. Von hier aus wird z.B. die Speicherverwaltung, das Power-Management oder auch das Kamera-Handling gesteuert.

#### Die Android Runtime

Die Android Runtime besteht aus Java-Core-Libraries und ist auf dem gleichen Layer wie die C-Libraries angegliedert. Diese Bibliotheken bilden zusammen einen funktionsfähigen Java Core, auf dem die Applikationen aufgebaut sind. All

<sup>1</sup> <http://developer.android.com/guide/basics/what-is-android.html>

diese Android Applikationen besitzen einen eigenen Prozess, welcher innerhalb der Dalvik Virtual Machine ausgeführt wird.

Die Dalvik Virtual Machine (DVM) ist eine für Android entwickelte Java Virtual Machine (JVM). Unter Dalvik können verschiedene Instanzen gleichzeitig laufen. So kann garantiert werden, dass mehrere Prozesse gleichzeitig ausgeführt werden können.

## Die C-Libraries

Der nächste Level des Software Stacks beinhaltet die verschiedenen Bibliotheken. Diese Libraries erteilen den verschiedenen Hardware-Komponenten Anweisungen, wie sie sich zu verhalten und Daten zu verarbeiten haben. Die Media Framework Library unterstützt zum Beispiel, das Aufnehmen und Abspielen von verschiedenen Audio-, Video- und Bild-Formaten.

## Application Framework

Das komplette Application Framework ist ein Programmiergerüst, welches den Rahmen für die Entwicklung von Android Applikationen bereitstellt. Es beinhaltet alle Basisfunktionen, wie zum Beispiel das Wechseln zwischen zwei Prozessen oder die Verarbeitung der GPS-Daten.

Alle Entwickler haben volle Zugriffsrechte auf dieses Framework. Das heisst, dass die Core Applikationen, die bereits beim Kauf eines Android installiert sind, nicht priorisiert werden.

Im Framework sind Services vordefiniert, welche von den Entwicklern direkt übernommen werden können. So kann zum Beispiel der Notification Manager verwendet werden, um Statusmeldungen oder sonstige Aktivitäten in der Benachrichtigungszeile anzuzeigen.

Die einzelnen Bestandteile des Application Frameworks sind in im nächsten Kapitel genauer erklärt.

## Applications

Die verschiedenen Applikationen, welche bereits beim Kauf eines Android Mobiltelefons installiert sind und Applikationen / Spiele, welche später heruntergeladen werden. Dies kann z.B. ein SMS-, E-Mail-, Browser-Programm sein.

### 3.3 Das Layout und die Activities

Mit Hilfe von Activities kann der Nutzer mit der Applikation agieren. Sie repräsentieren eine Bedienoberfläche und können verschiedene GUI-Elemente, wie z.B. Layouts oder auch Listen beinhalten. Eine Activity-Klasse erbt jeweils von der Klasse `Activity`. Falls der Nutzer eine Activity beendet, wird meistens eine neue gestartet. Dies kann mit einem Seitenwechsel einer Webseite verglichen werden. Beim Wechsel zwischen Activities wird der vorige jeweils auf einen History Stack gespeichert. Durch Klick auf den „Back“-Button kann der Nutzer immer auf zum vorherigen Screen zurückkehren.

### 3.4 Der Resource Manager<sup>1</sup>

Ressourcen sind Nicht-Code-Files, die in der Applikation verwendet werden. Dies können z.B. Bilder, Layout-Files oder auch String-Files sein. Da der *Resource Manager* die Logik von der Präsentation trennt, ist er ein zentraler Bestandteil von Android. Folgende Verzeichnisse sind im Ressourcen Manager eingegliedert:

- `/res/anim/`
- `/res/drawable`
- `/res/layout`
- `/res/values`

#### Inhalt von `/res/anim/`

Mit Android kann man Grafiken verschiedentlich manipulieren. Zum Beispiel wird das Foto vergrößert, verkleinert, der Alpha-Wert geändert. In diesem Verzeichnis werden die XML – Files, die diese Manipulation durchführen abgespeichert.

#### Inhalt von `/res/drawable/`

In diesem Verzeichnis werden alle im Programm verwendeten Bilder gespeichert. Falls die Applikation in mehreren Sprachen verfügbar sein soll, so kann die Ordnerstruktur...

- `res/drawable-de`
- `res/drawable-fr`
- `res/drawable-en`
- `etc.`

... verwendet werden.

---

<sup>1</sup> <http://developer.android.com/guide/topics/resources/resources-i18n.html>

## Inhalt von /res/layout/

Alle Layoutfiles, die in der Applikation verwendet werden, können hier abgespeichert werden. Wie man mit den Layouts arbeitet, wird im Kapitel 3.5 genauer erklärt.

## Inhalt von /res/values/

In diesem Verzeichnis werden alle Strings und Id's gespeichert. Falls eine Applikation Mehrsprachigkeit unterstützen soll, kann man dies sehr einfach realisieren. Für alle gewünschten Sprachen ist ein eigenes Verzeichnis anzulegen. Für die Deutsche Sprache z.B. /res/values-de, für Französisch /res/values-fr. Nun verwendet Android je nach Einstellung der Handysprache das dazugehörige values-Verzeichnis. Falls die gewählte Sprache nicht in der Applikation unterstützt wird, liest die Applikation den Inhalt von /res/values.

Üblicherweise sind im Verzeichnis /res/values(-xx) folgende Files vorhanden:

**Tabelle 2: Inhalt von /res/values**

arrays.xml	Alle String-Arrays werden hier definiert.
strings.xml	Jeder Textblock, der in der Applikation verwendet wird, sollte hier definiert werden. Diese Datei dient als Sprachfile.
colors.xml	Alle gewünschten Farben können hier definiert werden.

Der Inhalt von /res/values/ ist nicht strikt definiert. Es können auch andere .XML-Files gespeichert werden und anschliessend mit Hilfe der Android Resource Klasse „R“ ausgelesen und verwendet werden.

## Die Klasse „R“

Die „R-Klasse“ wird automatisch generiert und sollte nicht von Hand abgeändert werden. Bei jedem „Build“ der Applikation wird sie neu geschrieben. Diese Klasse stellt die Schnittstelle zwischen Nicht-Code-Files und der Logik dar.

Eine solche Klasse könnte z.B. folgendermassen aussehen:

Code-Abschnitt 3-1: Beispiel einer Android R. Klasse

```
1  public final class R {
2      public static final class string {
3          public static final int greeting=0x0204000e;
4          public static final int start button text=0x02040001;
5          public static final int submit_button_text=0x02040008;
6          public static final int main screen title=0x0204000a;
7      };
8      public static final class layout {
9          public static final int start_screen=0x02070000;
10         public static final int new user pane=0x02070001;
11         public static final int select_user_list=0x02070002;
12     };
13     public static final class drawable {
14         public static final int company_logo=0x02020005;
15         public static final int smiling_cat=0x02020006;
16         public static final int yellow_fade_background=0x02020007;
17         public static final int stretch button 1=0x02020008;
18     };
19 };
20 };
21 };
```

In diesem Beispiel sieht man, dass für alle Verzeichnisse in `/res/` eine private Klasse erstellt wird. Für das Verzeichnis `String`, das nur eine Datei (`strings.xml`) beinhaltet, wurde die Klasse `string` (Z. 2) generiert. All die eingetragenen Strings erhalten nun eine eindeutige Id. Der `start_button_text` bspw. erhält die Id `0x02040001`. Da alle Identifier `public` sind, kann nach dem Importieren der Klasse „R“ auf alle Ressourcen zugegriffen werden.

Um nun einen String auszulesen, kann man die Methoden `Resources.getString()` oder `Resources.getText()` aufrufen:

Zugriff auf eine String-Ressource

```
String myString = Resources.getText(R.strings.start_button_text);
```

Neben den applikationsabhängigen Ressourcen gibt es zudem noch System-Ressourcen, die in der Klasse `android.R` gespeichert sind. Der Zugriff auf diese Klasse erfolgt gleich wie bei den eigenen Ressourcen (`android.R.strings.start_button_text`). Da Doppeleinträge keinen Sinn machen würden und um den Android Guidelines zu entsprechen, sollte man, falls möglich, immer auf die Systemressourcen zugreifen. In diesen sind z.B. alle Systemabhängigen Icons oder Button-Texte (*ok*, *cancel*, etc.) gespeichert.

## 3.5 Das Layout Prinzip / Die Views

Views sind Sichten, welche das Layout von Android Activities repräsentieren. Es gibt viele verschiedene Layouts, welche der Entwickler wählen kann:

- LinearLayout
- RelativeLayout
- TableLayout
- DatePicker
- TimePicker
- Verschiedene Formulareinträge (Textboxes, Labels, Checkboxes u.v.m.)
- GridView
- Gallery
- TabWidget
- MapView
- WebView
- etc.

Diese verschiedenen Views können beliebig miteinander kombiniert werden. Ein umfassendes Tutorial zum Thema „Views“ kann unter folgenden Adresse abgerufen werden: <http://developer.android.com/guide/tutorials/views/index.html>

Das am häufigsten verwendete Layout ist das `LinearLayout`. Im folgenden Beispiel ist ein solches mit einer Textbox und einem Button dargestellt:

Code-Abschnitt 3-2: Beispiel eines LinearLayouts

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:orientation="vertical"
5     android:layout_width="fill_parent"
6     android:layout_height="fill_parent">
7
8     <TextView
9         android:id="@+id/name"
10        android:text="red"
11        android:layout_width="wrap_content"
12        android:layout_height="fill_parent"
13        android:layout_weight="1"/>
14
15    <Button
16        android:id="@+id/submit"
17        android:layout_width="fill_parent"
18        android:layout_height="wrap_content"
19        android:layout_alignParentBottom="true"
20        android:text="@string/submit_text" />
21 </LinearLayout>
```

Die Layout-Files sind gewöhnliche XML-Files mit Android spezifischen Tags. Um z.B. ein `LinearLayout` zu definieren ist das Tag `<LinearLayout>`



`</LinearLayout>` notwendig. Den verschiedenen XML-Tags können Attribute zugewiesen werden. Dies kann beispielsweise eine ID, ein Text oder auch die Grösse sein.

Um ein Layout einer Activity zuzuweisen ruft man in der Methode `onCreate()` Folgendes auf:

```
1 setContentView(R.layout.myLayoutName);
```

Einbindung eines Layouts

Nun sind alle Inhalte dieses Layouts in der entsprechenden Activity verfügbar. Eine `view` ruft man nun folgendermassen auf:

```
1 TextView tvName = (TextView) findViewById(R.id.name);
```

Zuweisen einer TextView

Man erstellt eine neue `view` (in diesem Fall eine `TextView`) und weist sie dem Layout-File zu. Dies erfolgt mit Hilfe der Methode `findViewById()`. Da diese Methode für alle Views gültig ist, muss das Resultat noch mittels `(TextView)` gecastet werden.

## 3.6 Das Manifest -File<sup>1</sup>

### Wozu das Manifest-File?

Jede Applikation verwendet ein sogenanntes Manifest-File. Plaziert wird diese Datei an der Wurzel des Android-Projektes und enthält wichtige Informationen über den Inhalt der Applikation.

Das Manifest-File...

- definiert das Java Package der Applikation (z.B. `com.android.hello`)
- beschreibt welche Activities eine Applikation beinhaltet
- beschreibt welche Zugriffsrechte die Applikation benötigt (z.B. Internet, GPS) und welche Rechte andere Applikationen benötigen um auf diese Applikation zuzugreifen.
- definiert welche Aktivität zu Beginn der Applikation gestartet wird.
- deklariert die minimale API-Version, welche die Applikation benötigt. Im Normalfall ist dies zur Zeit die Version 1.5

---

<sup>1</sup> <http://developer.android.com/guide/topics/manifest/manifest-intro.html>

## Der Aufbau eines Manifest-Files

Code-Abschnitt 3-3: Beispiel eines Manifest-Files

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.android.hello"
4     android:versionCode="1"
5     android:versionName="1.0.0">
6     <application android:icon="@drawable/icon"
7         android:label="@string/app_name">
8         <activity android:name=".HelloAndroid"
9             android:label="@string/app_name">
10             <intent-filter>
11                 <action android:name="android.intent.action.MAIN" />
12                 <category
13                     android:name="android.intent.category.LAUNCHER" />
14             </intent-filter>
15         </activity>
16         <activity
17             android:name=".activities.ActRegistration"></activity>
18         <activity
19             android:name=".activities.ActCamera"
20             android:screenOrientation="landscape"></activity>
21         <activity android:name=".activities.ActHistory"></activity>
22     </application>
23     <uses-permission
24         android:name="android.permission.INTERNET"></uses-permission>
25     <uses-permission
26         android:name="android.permission.CAMERA"></uses-permission>
27 </manifest>
```

Wie bei einem gewöhnlichen XML File, steht zu Beginn der Datei die XML-Deklaration. Der Inhalt umschliesst das Tag `<manifest></manifest>`. Die Package Angabe in Zeile 3 muss mit der reellen Package-Beschriftung im Projekt übereinstimmen. Dieses Package gilt als eindeutige Identifikation der Applikation.

Die Versionsangaben (Zeilen 4 - 5) sind für die Veröffentlichung der Applikation sehr wichtig. Sie ermöglichen das *Versioning* auf dem Android Market. Der Zweig `<application></application>` (Zeilen 6-22) definieren die verschiedenen Activities. Der Intent-Filter (Zeilen 10 - 14) zeigt auf, dass diese Activity beim Starten der Applikation angezeigt werden soll.

Falls eine Activity standardmässig im Querformat gestartet werden soll, so kann man dies mit Hilfe des Parameters `android:screenOrientation="landscape"` (Zeile 20) tun. Das Tag `<uses-permission></uses-permission>` (Zeilen 24 - 26) gibt an, welche Rechte diese Applikation verwendet. Im oben dargestellten Beispiel stellt sie eine Verbindung zum Internet und zur Kamera her. Diese Zugriffsrechte muss ein Nutzer akzeptieren bevor er die Software installiert.

## Die Regeln, die eingehalten werden müssen

- Das Tag `<manifest></manifest>` muss zwingend am Anfang der Datei stehen und darf nur einmal vorkommen. Alle anderen Tags müssen nicht unbedingt vorhanden sein.
- Die Reihenfolge innerhalb von Elementen spielt keine Rolle. Einzige Ausnahme ist dabei das Tag `<activity-alias></activity>`. Dieses muss nach `<activity> </activity>` stehen
- Ganz Allgemein gesagt, sind alle Attribute optional. Doch um einige Elemente korrekt anzuzeigen sind, werden bestimmte Attribute vorgeschrieben. Wann welche Attribute nötig sind, ist in der folgenden Dokumentation ersichtlich:

<http://developer.android.com/guide/topics/manifest/manifest-intro.html#filestruct>

- Ausser bei einigen Attributen im Element `<manifest></manifest>` steht vor allen der Präfix `android:` z.B bei `android:name`
- Falls mehr als ein Wert innerhalb eines Tags angegeben werden muss, so sind diese gut strukturiert untereinander zu platzieren.
- Ressourcen, die als Attribute angegeben werden müssen, werden wie folgt deklariert: `@[package:]type:name`
- Bei String Werten mit Charakteren die ein Backslash (`\`) benötigen, ist ein weiterer Backslash hinzuzufügen. z.B. „`\\n`“ für eine neue Linie )

## Intents<sup>1</sup>

Intents beschreiben die verschiedenen Aktionen, die eine Android Applikation ausführt. Zu Intents gehören Daten, meistens Activities und Aktionen. Das unten aufgeführte Beispiel zeigt, wie man mit Hilfe eines Intents eine neue Activity starten kann. Dieser Codeblock wird aus einer Activity aus ausgeführt.

Beispiel für die Nutzung eines Intents

```
1 Intent newActivity = new Intent(this,myActivity.class);
2 startActivity(newActivity);
```

Bei der Deklaration eines Intents gibt man die Daten an, welche manipuliert werden sollen. Alternativ können diese auch mit der Methode `.setClass()` angegeben werden.

---

<sup>1</sup> <http://developer.android.com/reference/android/content/Intent.html>

## 4 Die Android SDK

### 4.1 Einführung

Bevor man Applikationen auf einem Handy entwickelt, sollte man sich zuerst mit dem Emulator von Android vertraut machen.

### 4.2 Voraussetzungen<sup>1</sup>

Um die Android SDK zu nutzen sind folgende Voraussetzungen nötig:

- Windows XP 32-bit oder Windows Vista 32-bit / 64-bit
- oder Mac OS X 10.4.8 oder höher (x68)
- oder Linux (Getestet unter Linux Ubuntu Dapper Drake)
- Eclipse IDE (Europa oder Ganymede)
- Eclipse JDT plugin
- Eclipse Classic IDE Paket wird nicht unterstützt
- JDK 5 oder JDK 6
- Android Development Tools Plugin (optional)

### 4.3 Vorbereitungen

#### Anmerkung

Zwar ist die Verwendung der SDK ohne Eclipse Plugin ebenfalls möglich, doch dadurch wird die Entwicklung von Android Applikationen unnötig erschwert. Beispielsweise wird mit Hilfe des Plugins der Umgang mit dem Emulator von Android wesentlich erleichtert. Bei der Entwicklung von Android Applikationen ist Eclipse zur Standard-Entwicklungsumgebung herangereift.

Die Codebeispiele und Applikationen, welche für die Erstellung dieser Arbeit entwickelt wurden, sind alle mit Hilfe des Eclipse-Plugins entwickelt worden. Deshalb wird hier nur die Installation unter Eclipse vorgestellt.

Abbildung 3: Eclipse Plattform



Quelle: [www.eclipse.org](http://www.eclipse.org)

---

<sup>1</sup> [http://developer.android.com/sdk/1.5\\_r1/requirements.html](http://developer.android.com/sdk/1.5_r1/requirements.html)

## Installation von Eclipse

Bevor man die SDK Installation und Konfiguration durchführen kann, muss man zuerst das Eclipse Paket herunterladen und installieren. Die aktuelle Version der Entwicklungsumgebung kann unter <http://www.eclipse.org/downloads/> heruntergeladen werden. Danach ist die IDE für Java EE Developers Version zu wählen.

Für die Nutzung von Eclipse ist keine Installation nötig. Das heisst, dass man Eclipse in ein gewünschtes Verzeichnis platzieren kann. Es wird empfohlen, dass man das Eclipse Verzeichnis in „C:\Program Files\eclipse“ ablegt.

## 4.4 SDK-Installation<sup>1</sup>

### Download

Unter [http://developer.android.com/sdk/1.5\\_r1/index.html](http://developer.android.com/sdk/1.5_r1/index.html) kann man die SDK herunterladen. In der SDK sind die Development Tools, die eigentlichen Plattformen (1.1, 1.5), die Dokumentation, einige Beispiel-Anwendungen und Add-Ons enthalten. Die SDK muss nun an einen geeigneten Ort (z.B. C:\Program Files\eclipse\AndroidSDK) heruntergeladen und danach mittels Klick auf „unzip“ entpackt werden.

### Konfiguration für Eclipse 3.3 (Europa)

- Eclipse starten und den Reiter „“ wählen
- „Search for new features to install“ anwählen und auf Next klicken
- “New Remote Site” auswählen
- Geben Sie danach in der erscheinenden DialogBox folgenden Eintrag ein: <https://dl-ssl.google.com/android/eclipse/>\*
- Bestätigen Sie die Eingabe und klicken Sie auf „Finish“
- Im erschienen Dialogbox muss man nun die neuen Einträge „Android DDMS“ und „Android Development Tools“ auswählen und mit Hilfe des Next-Buttons bestätigen.
- Akzeptieren Sie die Nutzungsbedingungen und Installieren Sie die Plugins.

\* Falls Probleme auftreten, kann man auch alternativ *http* anstatt *https* wählen.

---

<sup>1</sup> [http://developer.android.com/sdk/1.5\\_r1/installing.html](http://developer.android.com/sdk/1.5_r1/installing.html)

## Konfiguration für Eclipse 3.4 (Ganymede)

- Eclipse starten und den Reiter „*Help -> Software Updates*“ wählen
- „*Available Software*“ Register anwählen und auf *Next* klicken
- Mittels Klick auf „*Add Site*“ die gewünschte Seite hinzufügen
- In der darauf erscheinenden DialogBox muss man folgenden Eintrag eingeben: <https://dl-ssl.google.com/android/eclipse/>\*
- Die Eingabe ist mittels Klick auf „*Ok*“ zu bestätigen.
- Im erschienenen Dialogbox muss man nun die neuen Einträge „*Android DDMS*“ und „*Android Development Tools*“ auswählen und mit Hilfe des Install-Buttons bestätigen.
- Nach der Bestätigung der Nutzungsbedingungen werden die Plugins installiert.

\* Falls Probleme auftreten, kann man auch alternativ *http* anstatt *https* wählen.

## Konfiguration der Android SDK

Nach dem Neustart der Eclipse Umgebung wählen Sie folgende Option aus: „*Window->Preferences*“. Im Register Android muss nun der Pfad der SDK angegeben werden. Nun sollte die SDK wunschgemäß funktionieren und mit der Entwicklung der ersten Android Applikation kann begonnen werden.

## 4.5 SDK-Tools

### Der Dalvik Debug Monitor Service (DDMS)<sup>1</sup>

Android Applikationen können nicht mit dem Standard-Debugtool von Eclipse getestet werden. Dazu hat Android ein eigenes Tool entwickelt und mit der SDK mitgeliefert. Der DDMS kann Informationen über Threads, Funkverbindungen und Heap Informationen liefern, kann jedoch auch Screenshots eines Gerätes aufnehmen.

Der DDMS agiert als „Mittelsmann“ zwischen der IDE und den verschiedenen Applikationen. Da jede Applikation von Android in einem separaten Prozess läuft, können diese einzeln aufgesplittet und getestet werden. Um beispielsweise Variable zu debuggen, ist die Verwendung der Klasse `Log` sehr nützlich. Wie man diese Klasse nutzt wird im Bereich „*LogCat*“ dieses Kapitels genauer beschrieben.

Geöffnet wird der DDMS mit Hilfe von Eclipse. Unter „*Windows/Open Perspective*“ wählt man die Option DDMS. Die Perspektive ist in zwei Bereiche gegliedert. Links sieht man die verschiedenen Geräte, welche angeschlossen sind und gerade laufen. Ebenfalls sind die geräteabhängigen Prozesse und Threads

---

<sup>1</sup> <http://developer.android.com/guide/developing/tools/ddms.html>

aufgelistet. Des Weiteren können hier verschiedene Simulationen durchgeführt werden. Es können z.B. eingehenden Anrufe oder empfangen SMS simuliert werden.

Auf der rechten Seite befindet sich der Hauptschirm. Dieser ist in verschiedene Register aufgegliedert. Zum einen sind hier die verschiedenen Status-Informationen angegeben, zum anderen Threads-, sowie Heaps-Informationen abgebildet.

Die Threads sind eingeteilt in verschiedene Datenfelder: Angezeigt werden können hier die verschiedenen Id's und der Namen, sowie auch der Zustand des aktuellen Threads.

**Tabelle 3: Thread Zustände**

Running	Der Thread läuft. Der Code zu diesem Thread wird ausgeführt
Sleeping	Die Methode <code>Thread.sleep()</code> wurde aufgerufen. Der Thread ruht.
Monitor	Der Thread wartet auf einen Monitor Lock um weitergeführt zu werden.
Wait	Der Thread wartet. Der Status wurde in <code>Object.wait()</code> definiert.
Native	In diesem Thread wird nativer Code ausgeführt.
Vmwait	Der Thread wartet auf eine Ressource der VM.
Zombie	Der Thread ist in einem „Sterbensprozess“. Er wird beendet.
Init	Der Thread wird gerade initialisiert. Normalerweise sollte man diesen Zustand nicht sehen.
Starting	Der Thread wird gerade gestartet. Normalerweise sollte man diesen Zustand nicht sehen.

Neben den ob genannten Register steht ein weiteres Features zur Verfügung: Der sogenannte File Explorer, in dem alle vorhandenen Dateien auf dem Emulator oder dem Gerät angezeigt werden.

Zudem wird hier das Logging-Tool „Logcat“ eingebunden. Sofern im Code angegeben, werden hier alle Aktionen *StackTrace*-mässig aufgelistet. Weitere Informationen zum Logcat-Tool werden im nächsten Bereich beschrieben.

## Das LogCat

Mit Hilfe dieses Tools können Variablen genau analysiert werden. Alle Errors und Warnungen werden hier aufgelistet. Der User kann den gesamten *StackTrace* betrachten und gegebenenfalls reagieren.

Um eigene Variablen zu überprüfen sollte man zu Beginn der Klasse eine Variable `TAG` definieren:

```
1 private static final String TAG = "MyActivity";
```

Definition einer Log-Variable

Durch die Nutzung der Klasse `Log`, können nun die verschiedenen Meldungen angezeigt werden. Variablen, die zu Debugzwecken ausgegeben werden sollen, können folgendermassen definiert:

```
1 Log.v(TAG, "index=" + i);
```

Methoden-Aufruf für LogCat

So wird die Variable `i` („index= 1“, „index = 2“ etc.) im Log angezeigt. Die Definition eines „Tags“ dient zur Filterung von Nachrichten. Da verhältnismässig viele Informationen im Log-Trace angezeigt werden, kann man bequem seine eigenen Variablen herausfiltern und in einem separaten Register anzeigen lassen.

Abbildung 4: Anzeige des LogCats

```
07-27 11:03... W 569 WindowManager No focus window, dropping: KeyEve...
07-27 11:03... I 569 WindowManager Ignoring HOME; there's a ringing...
07-27 11:03... I 569 ActivityManager Displayed activity: ch.rfidcenter...
07-27 11:03... I 569 ActivityManager Starting activity: Intent { actio...
07-27 11:03... D 745 ch.rfidcenter.code... french
07-27 11:03... W 610 IInputConnectionWr... showStatusIcon on inactive InputC...
07-27 11:03... I 569 ActivityManager Displayed activity: ch.rfidcenter...
07-27 11:03... I 569 ActivityManager Starting activity: Intent { comp=...
07-27 11:03... I 569 ActivityManager Displayed activity: ch.rfidcenter...
07-27 11:03... I 569 ActivityManager Starting activity: Intent { comp=...
07-27 11:03... D 745 ActMenu 6
07-27 11:03... D 745 ch.rfidcenter.code... french
07-27 11:03... I 569 ActivityManager Displayed activity: ch.rfidcenter...
07-27 11:03... D 745 dalvikvm GC freed 4043 objects / 213120 by...
07-27 11:03... D 608 dalvikvm GC freed 10148 objects / 582336 b...
07-27 11:03... D 610 dalvikvm GC freed 1067 objects / 54096 byt...
07-27 11:03... W 745 KeyCharacterMap No keyboard for id 0
07-27 11:03... W 745 KeyCharacterMap Using default keymap: /system/usr...
07-27 11:03... I 569 ActivityManager Starting activity: Intent { comp=...
```

Quelle: Stefan Theler

Die Einträge werden jeweils mit dem Datum, der Zeit, dem Typen (E für Error, D für Debug...), dem Tag und der Meldung angezeigt. Es gibt 6 verschiedene Debugtypen, welche von der Klasse `Log` unterstützt werden:

- `ASSERT (Log.a())`
- `DEBUG (Log.d())`
- `ERROR (Log.e())`
- `INFO (Log.i())`
- `VERBOSE (Log.v())`
- `WARN (Log.w())`

Nach diesen verschiedenen Typen kann im *LogCat* ebenfalls gefiltert werden.

Das *LogCat* wird immer ausgeführt, auch wenn die Applikation nicht im Debugmodus gestartet wird. Falls ein Gerät mittels USB mit einem Rechner verbunden und das DDMS gestartet ist, kann der *StackTrace* ohne weiteres abgerufen werden.



## Die Traceview Analyse

Das Traceview ist eine grafische Ausmessung von Log-Files. *Traceview* kann die Performance von verschiedenen Methoden und Klassen analysieren und so Schwachstellen in der Applikation aufzeigen.

Ein *Traceview*-File erstellt man folgendermassen:

Code-Abschnitt 4-1: Beispiel eines TraceView Trackings

```
1 // start tracing to "/sdcard/myTrackingFile.trace"
2 Debug.startMethodTracing("myTrackingFile");
3
4 /*
5 Place your Code here!
6 */
7
8 // stop tracing
9 Debug.stopMethodTracing();
```

Die Tracking-Files werden auf die interne SD-Card gespeichert. Um diese nun auf den Rechner zu portieren ist ein USB Kabel und die `adb`-Shell (Erklärung `adb` im nächsten Bereich) notwendig.

Kopieren des Tracking-Files

```
C:\Path_to_SDK\tools\>adb pull /sdcard/myTrackingFile.trace
/C:\myPathToLocation
```

Mit diesem Kommando kann das File lokal abgespeichert werden. Um nun das *TraceView*-Fenster zu öffnen, wird wieder die Kommandozeile verwendet.

Öffnen eines Tracking-Files

```
C:\Path to SDK\tools\>traceview
C:\myPathToTrackingFile\myTrackingFile.trace
```

Mit *TraceView* ist klar ersichtlich, welche Methoden wie viel Zeit in Anspruch nehmen. In der CodeCave Applikation konnte man dank dieses Tools feststellen, dass einige wichtige Methoden sehr viel Zeit für die Ausführung verwenden und so konnte man reagieren und die Programmierung abändern. Genauere Informationen zu diesem Beispiel, zu dieser Problemstellung sind im Kapitel 6.5 genauer beschrieben.

## Die Android Debug Bridge

Die Android Debug Bridge (`adb`) ist eine Kommandozeilen-Applikation, die Android verständliche Anweisungen beinhaltet. Mit Hilfe der `adb`-Konsole kann man weitere Android Tools aufrufen, z.B. den DDMS.

Des Weiteren kann die Android Debug Bridge Manipulationen direkt auf dem Emulator oder dem Gerät ausführen. Syntax sieht diesbezüglich wie folgt aus:

Aufrufen von adb-Befehlen

```
C:\Path_To_SDK\tools\>adb [-d|-e|-s <serialNumber>] <command>
```

Über die adb Konsole ist es einem Entwickler gestattet, Applikation auf einen Emulator oder einem Android-fähigen Smartphone zu installieren. Dies geschieht mit Hilfe des Befehls `adb install`. Um eine Datei von einem Emulator/Gerät zu kopieren verwendet man den Befehl `adb pull` und umgekehrt den Befehl `adb push`.

Eine vollständige Liste, welche Kommandos unterstützt werden kann hier abgerufen werden:

<http://developer.android.com/guide/developing/tools/adb.html#commandsummary>  
(Englisch, Letzter Aufruf: 27.07.2009).

## 5 Einführung in die Codelcare Applikation

### 5.1 Allgemeines<sup>1</sup>

Mit mehr als 3 Milliarden Handynutzern weltweit, ist das Mobiltelefon zu einem wichtigen Bestandteil in der Kommunikation und Integration zwischen Unternehmen und Verbrauchern geworden. Heutzutage können Handys Barcodes lesen und sie können sich auch mit dem Internet verbinden. CodeIcare nutzt diese beiden Technologien um Produktinformationen benutzerfreundlich zu präsentieren.

Durch das Fotografieren eines Barcodes auf einem Arzneimittel oder auch einem anderen beliebigen Produkt, können Informationen aus dem Internet geholt und in der Applikation dargestellt werden.

### 5.2 Einführung in die Thematik der Barcodes

#### Was ist ein Barcode

Man unterscheidet im Groben zwei Arten von Barcodes. Zum einen die linearen und zum anderen die 2D-Barcodes. Die Strichcodes sind mit verschiedenen breiten Balken ausgestattet und ergeben eine eindeutige Ziffer. Im Gegensatz dazu, kann ein 2D-Barcode, eine Datenmatrize viel mehr Informationen abspeichern. Alle Barcodes enthalten ein Start- und Stopzeichen und sogenannte Quit-Zones. Letzteres sind Zwischenräume, welche keine Daten enthalten, sondern nur als Barcode-Begrenzung gelten. Durch die Quit-Zones, welche nach dem Startzeichen und vor dem Stopzeichen angebracht sind, wird der Barcode strikte abgetrennt. Das Startzeichen gibt dem Barcodescanner die Information, in welche Richtung gescannt wird. Das Stopzeichen gibt an, dass der Barcode hier nun zu Ende ist.

#### Der Unterschied zwischen Barcode und RFID-Tag<sup>2</sup>

Sowohl Barcodes wie auch RFID-Tags sind dazu da um Produkte eindeutig zu identifizieren. Doch damit enden bereits die Gemeinsamkeiten. Im Gegensatz zu einem Barcode, muss das Lesegerät bei einem Scan nicht unmittelbar vor das Produkt gehalten werden. RFID-Tags behalten ihre Informationen so lange zurück bis ein Lesegerät, die Tags mittels Funkverbindung auffordert, ihren Inhalt preis zu geben. Ein solches Lesegerät kann durchaus auch einige Meter weiter entfernt stehen. Des Weiteren lassen sich beliebige, viel genauere Informationen auf ein RFID-Tag schreiben und später wieder modifizieren. Ist

---

<sup>1</sup> <http://www.rfidcenter.ch/InternetDesObjets/contexte.html>

<sup>2</sup> <http://www.rfid-journal.de/barcode.html>

ein Barcode einmal gesetzt, so kann dieser nicht mehr abgeändert werden. Bei der RFID-Technologie kann jedes Produkt genau gekennzeichnet werden. Dies ist bei herkömmlichen Barcodes nicht möglich. Einzig bestimmte Produktreihen können identifiziert werden.

Ein weiterer Punkt, welcher für die RFID-Technologie spricht, ist die Anfälligkeit von Barcodes auf Unlesbarkeit bei Verschmutzung oder z.B. Sonneneinstrahlung. Ein Barcode ist nur schwer zu lesen, wenn die Oberfläche glatt ist und die Sonne gespiegelt wird. Auch die Krümmung (z.B. bei PET-Flaschen) kann einen Einfluss haben.

Die RFID Technologie wurde schon lange als Nachfolger der herkömmlichen Barcodes gehandelt. Doch bislang ist die „alte“ Technik immer noch weit verbreiteter. Dies vor allem auf Grund der zusätzlichen Kosten, die bei der RFID-Technologie anfallen. Zudem braucht es einen speziellen Reader oder Handtyp, um RFID-Tags zu lesen. Barcodes können auf allen Handys mit Kamera gelesen werden.

Deshalb kann es bei bestimmten Verwendungszwecken durchaus Sinn machen auch weiterhin auf die RFID-Technologie zu verzichten.

## Die Unterschiede zwischen Strichcode und Datamatrix

Abbildung 5: Beispiel EAN und Datamatrix



Quelle: [www.wikipedia.org](http://www.wikipedia.org)

2D-Barcodes (Datenmatrizen) können, im Gegensatz zu Strichcodes nicht nur von links nach rechts gelesen werden. Zusätzlich ist eine Abtastung von oben nach unten möglich. Dies zeigt klar auf, dass bei 2D-Codes die Dichte der Informationen wesentlich höher ist als

bei einem linearen Strichcode. In einer Datenmatrize werden die Inhalte innerhalb von Blocken gleichmässig codiert. Sowohl bei linearen Strichcodes, wie auch bei 2D-Codes gibt es verschiedene Ausführungen. Auf alle einzugehen, würde den Rahmen dieser Bachelorarbeit sprengen. Die Applikation „CodeIcare“ unterstützt das gängige 1D Format „European Article Number“ (EAN) und die Datenmatrizen. Zu einem späteren Zeitpunkt soll auch die Decodierung von QR-Codes möglich. QR-Codes sind in die Kategorie der 2D-Codes einzuteilen und ähneln Datenmatrizen ziemlich stark. Rein optisch unterscheiden sich QR-Codes und Datenmatrizen einzig in den 3 Begrenzungsmarkierungen in den Ecken des QR-Codes.

## Warum Barcodeerkennung?

Schon seit vielen Jahren werden Barcodes an Produkten angebracht um sie zu identifizieren. Durch das einscannen der Barcodes erübrigt sich das mühsame

eintippen von Preisen eines Produktes. Grundsätzlich ist ein Barcode nichts anders als eine digitale Darstellung einer Identifikationsnummer. Zwar wäre es möglich diese Nummer einzutippen und mit einer Datenbank zu vergleichen, doch dies kann keinem Kunden zugemutet werden. Des Weiteren wäre die Fehlerquote viel zu hoch.

Dadurch macht es Sinn Produktinformationen bequem mit Hilfe eines Scanners anzuzeigen und weiter zu verarbeiten. Sei dies in Lebensmittel-, Warenhäuser oder auch zur privaten Nutzung.

### 5.3 Einführungsbeispiel

Abbildung 6: Einführungsbeispiel



Quelle: [www.springwater.nl](http://www.springwater.nl)

Auf einer Flasche San Pellegrino ist eine Datenmatrix (vergleichbar mit einem Barcode) angebracht. In dieser Datenmatrize ist nun ein eindeutig definierter Identifikationscode gespeichert. Dieser Code ermöglicht es, dieses Produkt von anderen zu unterscheiden. Im Falle der San Pellegrino Flasche wäre der Code folgender: 010764012572013621800000011

Codelcare fotografiert nun mit Hilfe der der integrierten Kamera des Mobiltelefons diese Datamatrix und generiert aus dem Bild die eindeutige Identifikationsnummer.

Diese Nummer wird anschliessend automatisch an einen Server des Icare Instituts übermittelt. Die Web-Applikation vergleicht den Code mit der Datenbank und holt so die Informationen zum gewünschten Produkt.

Falls gewünscht, kann der Nutzer seine Produkte in einem Verlauf speichern und so jederzeit wieder abrufen.

Damit die Informationen auf dem Mobiltelefon angezeigt werden, sendet der Server eine URL zurück. Diese URL wird mit dem internen Browser des Mobiltelefons aufgerufen und die Informationen werden so angezeigt.

## 5.4 Das Prinzip des Scannens

Abbildung 7: Das Prinzip der Verarbeitung



Quelle: Gestaltung Stefan Theler

Der Nutzer startet das Programm „codeIcare“ auf seinem Handy. Die Kamera Aktivität öffnet sich und er muss sein Handy vor einen Barcode oder vor eine Datenmatrize gehalten, so dass die Applikation automatisch einen Autofokus ausführen und den Barcode fotografieren kann.

Das Programm versucht nun den Code zu identifizieren und zu decodieren. Falls ein Code aus dem Bild erkannt wurde, erscheint eine Information, ob eine Internetverbindung hergestellt werden soll. Falls der Nutzer diese Meldung

bestätigt, wird eine Anfrage an den Server des Institut Icares gesendet.

Der Server kann folgende Antworten liefern:

- Der Code wird gefunden und die URL wird zurückgeschickt.
- Bei einem Aktivierungscode wird der Benutzer auf dem Server registriert und aktiviert. Der Nutzer wird informiert.
- Der Code wurde nicht gefunden. Der Nutzer erhält eine entsprechende Meldung.
- Der Benutzer ist nicht registriert und somit kann er keine Produktinformationen einholen. In diesem Fall erhält er eine entsprechende Meldung.
- Dem Benutzer wird ein neuer Service hinzugefügt und wird entsprechend informiert.

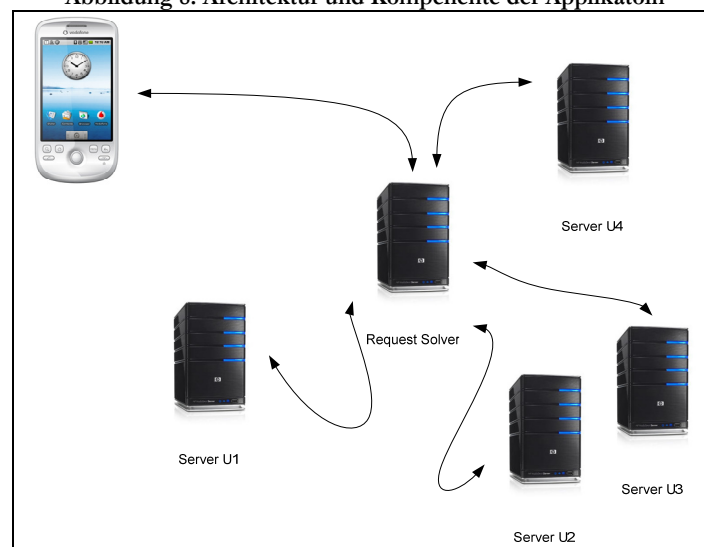
Ein Flussdiagramm, welches das Prinzip des Scannens verdeutlicht ist im Anhang beigelegt.

## 5.5 Die Architektur der Applikation

### Voraussetzungen / Hardware

Um diese Applikation zu nutzen ist ein Android fähiges Mobiltelefon nötig. Zudem wird ein Server des RFID-Centers, ein sogenannter *Request Solver* verwendet. Dieser Resolver leitet die Anfrage des Kunden an verschiedene Datenquellen weiter. Die Datenquellen können einerseits innerhalb von-Centers sein, oder auch externe, wie z.b. GEPiR von GS1. Interne Daten werden direkt als URL zurückgegeben, Externe hingegen werden von der Produktwebseite geholt und so dem Nutzer präsentiert.

Abbildung 8: Architektur und Komponente der Applikation



Quelle: Gestaltung Stefan Theler

Da die Applikation auch für andere Mobilfunktypen (iPhone, Java ME Version) angeboten werden, wurde der Serverteil bereits vollständig implementiert. In dieser Bachelorarbeit musste nur die Handyapplikation und die Verbindung zum Server entwickelt werden.

### Die Bestandteile der Applikation

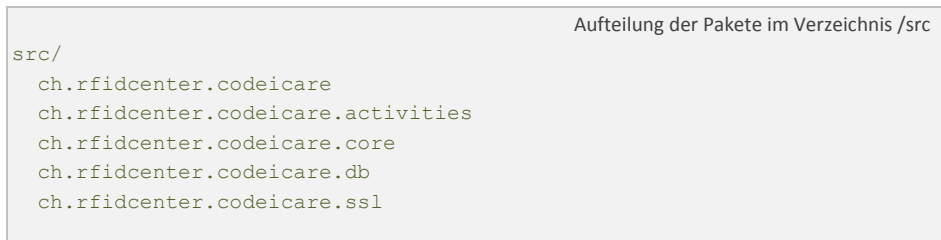
Die Applikation ist in 6 Bestandteile gegliedert:

- Das Scannen eines Codes
- Das manuelle Eingeben eines Codes
- Informationen über das RFID-Center
- Informationen über die Applikation
- Konfiguration
- Verlauf

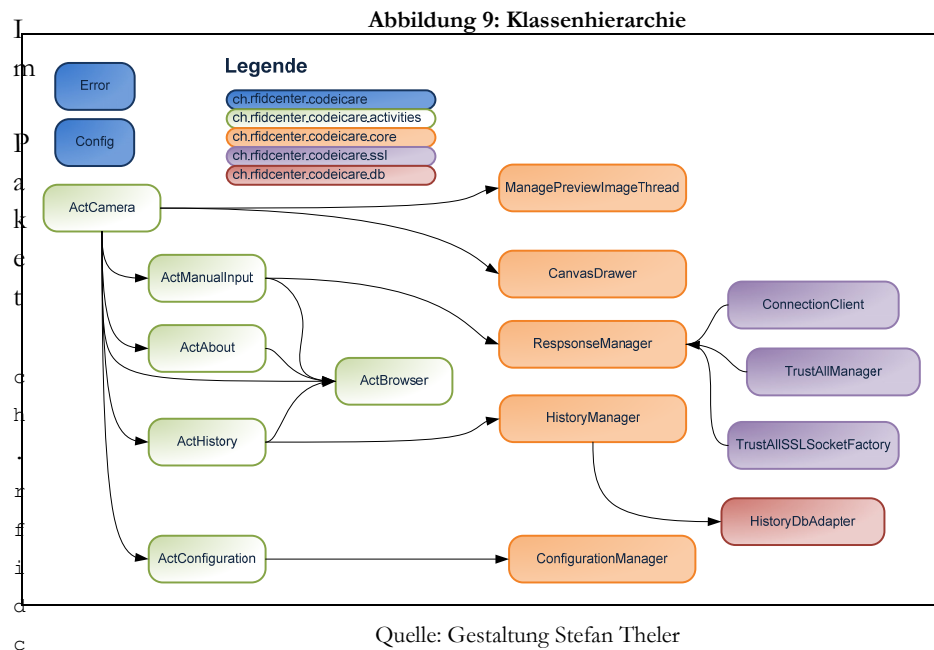
Der Hauptschwerpunkt der Applikation liegt beim Scannen eines Barcodes mit Hilfe der integrierten Kamera. Die Informationen über das RFID-Center stellen lediglich ein Link auf die Webseite von CodeIcare dar. Im Bereich Konfiguration kann der Nutzer seine Sprache einstellen und wählen, ob er den Verlauf nutzen möchte. Im Bereich Verlauf sind, falls eingestellt, alle vorgängig gescannten Produkte aufgelistet.

## Die Klassenhierarchie

Die Grundstruktur bleibt bei diesem Projekt dieselbe wie bei anderen Android Projekten. Die Struktur innerhalb der Source-Files gliedert sich wie folgt:



In der Grafik (Grossansicht in Anhang) sieht man die Anordnung der Klassen und deren Zusammenhang.



enter.codeicare befinden sich alle allgemeinen Klassen, wie zum Beispiel die Error oder Config Klasse. Im Paket ch.rfidcenter.codeicare.activities befinden sich alle Activities, welche in der Applikation verwendet werden. Die Logik wurde so gut wie möglich aus den Activities ausgegliedert und in sogenannten Manager-Klassen implementiert. Diese werden dann in den



Activities deklariert und initialisiert. Im Core-Paket sind alle wichtigen Klassen, welche die Logik der Applikation darstellt. Von hier aus wird unter anderem die Kamera gesteuert, die Konfiguration ausgelesen und den Verlauf verwaltet.

Das Paket `ch.rfidcenter.codeicare.db` beinhaltet alle Klassen, welche einen Datenbank Adapter darstellen, also eine Verbindung zur SQLite Datenbank aufbauen müssen. Da die Applikation nur bei der Nutzung des Verlaufes eine Datenbankanbindung verwendet, besitzt dieses Package nur eine Klasse.

Alle Internet-Verbindungsoptionen, welche auf SSL basieren, sind im Package `ch.rfidcenter.codeicare.ssl` enthalten. Da die Server des RFID-Centers bei der Erstellung der Applikation SSL noch nicht unterstützen, konnte die Funktionsfähigkeit der Klassen in diesem Paket nicht getestet werden.

## Die Datenbank

Unter Android gibt es 3 Möglichkeiten Daten persistent zu speichern<sup>1</sup>. Eine dieser Varianten ist das Speichern in Datenbanken. Als relationales Datenbanksystem setzt Android SQLite ein. Dieses System wird bereits in anderen Handy-Betriebssystemen, wie zum Beispiel im iPhone oder Symbian OS eingesetzt. Auch der Internet-Browser Mozilla Firefox nutzt ab Version 3 SQLite, um lokale Inhalte (Bookmarks, etc.) abzuspeichern.

Eine Datenbank unter Android ist privat und nur in dieser Applikation zu nutzen. Es sei denn man nutzt sogenannte `ContentProviders`<sup>2</sup>, welche ein Austausch von Daten von Programmen ermöglicht.

Wie bereits erwähnt, besteht die Datenbank von CodeIcare lediglich aus einer Tabelle. Nur bei der Nutzung des Verlaufes ist eine Datenbankanbindung nötig.

Die Tabelle `codeicare_history` besteht aus folgenden Felder:

- `_id`
- `url`
- `code`
- `tstamp`

Was die einzelnen Felder bedeuten, sowie weitere Informationen zum History-Handling können im Kapitel 6.7 nachgelesen werden.

---

<sup>1</sup> <http://developer.android.com/guide/topics/data/data-storage.html>

<sup>2</sup> <http://developer.android.com/guide/topics/providers/content-providers.html>

## 6 Technische Details zur Applikation

### 6.1 Menuhandling

Abbildung 10: Hauptmenü der Applikation



Quelle: Stefan Theler

Bei Android gibt es Richtlinien, die möglichst einzuhalten sind. Unter diesen ist auch die Weisung, dass man die offiziellen Menüs verwenden soll. Ein Android Menu zu implementieren ist sehr einfach. In einer bestehenden Activity überschreibt man einfach die Methode `onCreateOptionsMenu(Menu menu)` und füllt dieses mit Daten.

Code-Abschnitt 6-1: Erstellen eines Menüs

```
1 public boolean onCreateOptionsMenu(Menu menu) {  
2     super.onCreateOptionsMenu(menu);  
3  
4     MenuItem mnuManual = menu.add(0, 2, 1,  
5         R.string.manualinput_menu_title);  
6     mnuManual.setIcon(android.R.drawable.ic_menu_edit);  
7  
8     MenuItem mnuHistory = menu.add(0, 3, 2,  
9         R.string.history_menu_title);  
10    mnuHistory.setIcon(android.R.drawable.ic_menu_agenda);  
11  
12    SubMenu smnuMore = menu.addSubMenu(0, 4, 3, "More");  
13    smnuMore.setIcon(android.R.drawable.ic_menu_more);  
14  
15    smnuMore.add(0, 5, 0, R.string.conf_menu_title);  
16    smnuMore.add(0, 6, 1, R.string.about_menu_title);  
17    smnuMore.add(0, 7, 2, "Help");  
18  
19    return super.onCreateOptionsMenu(menu);  
20 }
```

Aus ActCamera.java, Methode `onCreateOptionsMenu()`

Menu-Einträge widerspiegeln Objekte der Klasse `MenuItem` und werden dem Menu hinzugefügt. Falls ein Untermenü erstellt werden muss, so können

Objekte der Klasse `SubMenu` implementiert werden (Zeilen 12 – 14). Damit man bei einem Klick-Event weiss welcher Eintrag gemeint ist, muss jeweils eine ID definiert werden. Diese ID wird bei den Methodenaufrufen `add()` und `addSubMenu()` jeweils mit dem zweiten Übergabeparamter mitgegeben.

Damit nach einem Klick auf ein Event ausgeführt wird, muss die Methode `onOptionsItemSelected(MenuItem item)` überschrieben werden. Mit Hilfe von `Switch`-Statements kann dann die entsprechende ID abgerufen und die gewünschte Activity gestartet werden.

Nach einem Klick auf einen Menü-Eintrag wird in CodeIcare Applikation immer eine neue Activity gestartet. Wie man allgemein eine neue Activity startet wird in den folgenden Programmcodezeilen untersucht.

```
1 Intent myActivity = new Intent(this, ActivityClass.class);
2 startActivity(myActivity);
```

Allgemeiner Aufruf einer Activity

Zuerst wird ein neues Intent mit der zu öffnenden Klasse erstellt. In der aktuellen Activity muss die Methode `startActivity()` mit dem erstellten Intent geöffnet werden. Die Activity wird danach automatisch gestartet.

Damit die Daten in der alten Activity nicht verloren gehen, wird dieser Zustand gespeichert und in einen sogenannten Ruhe-Modus gehalten. Bei einem Klick auf den Zurück-Button wird diese Aktivität wieder aktiviert und kann weiter verwendet werden.

## 6.2 Das Usermanagement

### Die Registrierung

Um die Applikation vollumfänglich zu nutzen ist eine Registrierung auf der Webseite <http://www.codeicare.ch> notwendig. Hat ein Benutzer die Applikation heruntergeladen, ist diese noch nicht aktiviert. Er kann zwar Codes scannen, die Informationen werden jedoch nicht angezeigt.

### Die Aktivierung

Nach der Registrierung erhält muss der Benutzer die Applikation aktivieren. Um zu überprüfen ob die Software bereits aktiviert ist, kann im Menu „Über uns“ nachgeschaut werden. Falls der Text „*Der Benutzer ist nicht registriert!*“ erscheint ist die Applikation noch nicht aktiviert.

Um dies zu tun muss sich der Benutzer mit seinem Konto anmelden und sich einen Aktivierungscode generieren lassen „*Ask a new activation code*“. Dieser wird mit der Software gescannt und die Applikation wird automatisch registriert. Der

Benutzer erhält eine Bestätigungsmeldung. Wie die Decodierung des Codes von statten geht, wird im Kapitel 6.3 genauer beschrieben.

Des Weiteren wird bei der Aktivierung ein Textfile erzeugt, in welchem die eindeutige Identifikationsnummer des Benutzers gespeichert wird. Diese Nummer wird bei jeder Produkteabfrage mitgeschickt.

Code-Abschnitt 6-2: Erzeugung einer UserId in User.java

```
1 public boolean setId(String id) {
2     FileOutputStream file;
3     try {
4         // Open the file
5         file = mContext.openFileOutput(
6             FILE_NAME,
7             Context.MODE_PRIVATE);
8         OutputStreamWriter writer =
9             new OutputStreamWriter(file);
10
11         // Write the string into the file
12         try {
13
14             writer.write(id);
15             writer.flush();
16             writer.close();
17
18         } catch (IOException e) {
19             Error.showErrorAsToast(mContext,
20                 R.string.eFileIOError);
21             e.printStackTrace();
22             return false;
23         }
24
25     } catch (FileNotFoundException e1) {
26         Error.showErrorAsToast(mContext,
27             R.string.eFileNotFoundError);
28         return false;
29     }
30
31     return true;
32
33 }
```

Aus User.java, Methode setId()

Das File wird `codeicare.id` genannt und wird direkt im Handy unter `/data/data/ch.rfidcenter.codeicare/files` gespeichert Zeilen (5 – 7). Der Parameter `Context.MODE_PRIVATE` bedeutet, dass die Datei nur von diesem Programm gelesen werden kann. Dies ist ein zusätzlicher Sicherheitsmechanismus

In der Zeile 14 wird die Identifikationsnummer in das File gespeichert. Diese Methode ist sehr ähnlich dem Speichern eines Textfiles in Java SE. Falls irgendwelche Exceptions ausgegeben müssen, werden diese aus der `Error`-Klasse erstellt und angezeigt. Die `Error`-Klasse wird im Kapitel 6.3 genauer beschrieben.

Falls der Benutzer sein Konto noch einmal aktivieren will, erscheint ein Fehler, dass der Code zu keinem Service gehört. Eine mehrmalige Aktivierung ist also nicht möglich und würde auch keinen Sinn machen.

## Die Verwendung der UserId

Wie bereits erwähnt, wird die UserId bei jedem Scanvorgang an den Server des RFID-Centers geschickt. Die UserId wird verwendet, um die verschiedenen Dienste eines Benutzers abzurufen. Nicht jeder Benutzer darf alle Informationen angezeigt bekommen.

Code-Abschnitt 6-3: Auslesen einer UserId

```
1 public String getId() {
2
3     FileInputStream file;
4     String readedString = "0";
5
6     try {
7
8         //Open the file
9         file = mContext.openFileInput(FILE_NAME);
10        InputStreamReader isr = new InputStreamReader(
11            file,
12            Charset.forName("UTF-8"));
13
14        // Get ready for Reading
15        BufferedReader br = new BufferedReader(isr );
16
17        // read file
18        readedString = br.readLine();
19
20    } catch (FileNotFoundException e) {
21        Error.showErrorAsToast(
22            mContext,
23            R.string.eFileNotFoundError);
24    } catch (IOException e) {
25        Error.showErrorAsToast(
26            mContext,
27            R.string.eFileIOError);
28    }
29    return readedString;
30 }
```

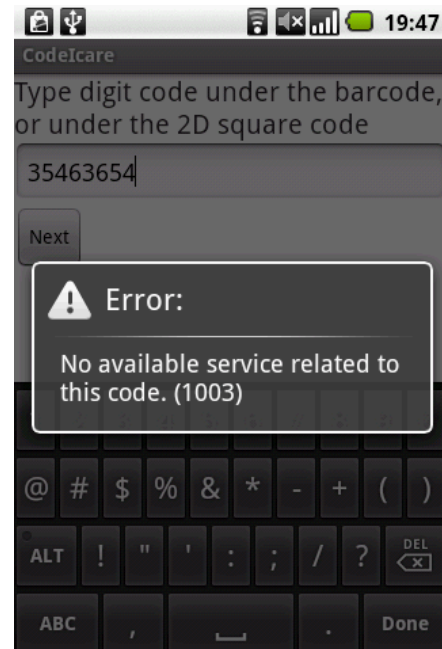
Aus User.java, Methode getId()

Um die UserId des Benutzers zu erhalten muss das File ausgelesen werden. Dies geschieht mit einem Standard `FileInputStream` der mit dem Charset `UTF-8` verwendet wird. Die Zeile 18 zeigt auf, wie eine Zeile ausgelesen werden kann. Der erhaltene String wird in Zeile 29 zurückgegeben.

## 6.3 Das Error-Handling

Die Applikation sollte normalerweise stabil laufen, doch es können immer wieder Fehler auftreten. Damit die Applikation möglichst wenig abstürzt wurde ein Fehlerhandling implementiert um dem Benutzer anzuzeigen, was genau falsch gelaufen ist. Die Fehlermeldungen werden in den Exceptions implementiert und ausgegeben. Es gibt zwei verschiedene Typen von Fehlermeldungen. Zum einen wird dies mit einem „Toast“ dargestellt und zu anderen mit einer „AlertDialog“. Ein „Toast“ ist eine Nachricht, welche für kurze Zeit angezeigt wird und danach von selbst verschwindet. Es ist der einfachste Weg, um einem Benutzer eine kleine Nachricht zukommen zu lassen.

Abbildung 11: Fehlermeldung



Quelle: Screenshot HTC Magic

Eine Toast-Meldung wird folgendermassen generiert:

```
Code-Abschnitt 6-4: Erzeugen eines Toast-Errors
1 public static void dieErrorAsToast(Context context, int id) {
2     Toast.makeText(
3         context,
4         context.getString(R.string.eLabel) + " " +
5             context.getString(id),
6         Toast.LENGTH_LONG).show();
7     Log.e(TAG, context.getString(id));
8 }
```

Aus Error.java, Methode dieErrorAsToast()

Ein Toast wird mit der Methode `makeText()` (Zeile 2 – 6) generiert und mit `show()` angezeigt (Zeile 6). Wichtig ist, dass jeweils ein Context mitgegeben wird, so der Toast, in welcher Activity er erscheinen soll. Der zweite Übergabeparameter stellt die Id der Fehlermeldung dar, diese wird dann mittels der String-Resource ausgelesen und angezeigt. Des Weiteren muss die Anzeigedauer angegeben werden. Android hat bereits Standardzeiten definiert, welche `LENGTH_LONG` und `LENGTH_SHORT` genannt werden. Falls ein Entwickler eine manuelle Dauer eingeben will, kann er dies selbstverständlich auch tun.

Bei schwerwiegenden Fehler, reicht eine wieder verschwindende Nachricht nicht aus. Der Benutzer muss mit einem Klick die Meldung bestätigen. Deshalb wurde ein `AlertDialog` implementiert, der dem Benutzer die Meldung anzeigt.

Code-Abschnitt 6-5: Erzeugen eines `AlertDialog-Errors`

```
1 public static void dieErrorAsDialog(Context context, int id) {  
2     Dialog dialog = new AlertDialog.Builder(context)  
3         .setIcon(android.R.drawable.ic_dialog_alert)  
4         .setTitle(context.getString(R.string.eLabel))  
5         .setMessage(context.getString(id)).create();  
6  
7     dialog.show();  
8 }
```

Aus `Error.java`, Methode `dieErrorAsDialog()`

Aus demselben Grund wie bei der Toast-Meldung wird ein `Context` und die `Id` mitgegeben. Im Gegensatz zum Toast, muss eine `AlertDialogBox` mit einem Klick bestätigt werden. Ein Icon bekräftigt den Eindruck des Fehlers und kann beliebig gewählt werden (Zeile 3). Ein Beispiel einer Dialog-Fehlermeldung ist in der Abbildung 11 gezeigt.

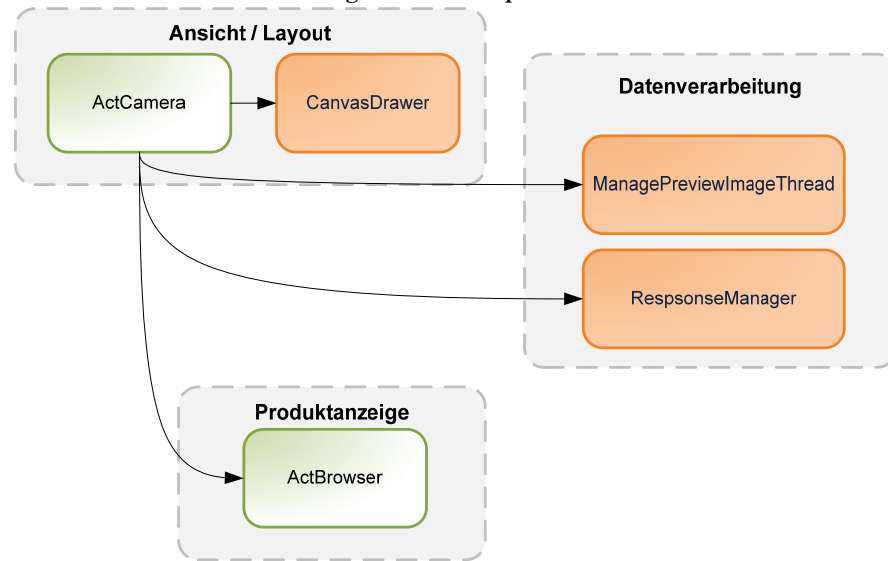
## 6.4 Das Barcode-Handling

### Allgemein

Dieser Teil ist das Kernstück der Applikation. Sie liest Barcodes ein und decodiert diese mit Hilfe einer nativen C Funktion. Zuerst wurde der gesamte Decoding-Bereich in Java programmiert. Doch schnell stellte sich heraus, dass Java für diese Aufgabe viel zu langsam arbeitet. Darum musste man rasch eine Alternative suchen. Der Einbau von C++ Funktionen stellte eine Möglichkeit dar. Diese Funktionen könne mit Hilfe von JNI (Java Native Interface) aufgerufen und verarbeitet werden. Man greift auf eine sogenannte „native Funktion“ zu und holt so Informationen aus der externen C++-Funktion. Mehr Informationen zum nativen C-Decoder erhält man im Kapitel 6.5.

## Das Prinzip

Abbildung 12: Das Prinzip des Scannens



Quelle: Gestaltung Stefan Theler

Zum Einlesen, Verarbeiten des Barcodes an den Decoder, sowie das Anzeigen des Produktes werden total fünf verschiedenen Klassen verwendet. Zum einem ist eine Activity nötig, zum anderen die Klasse `BarcodeManager`, die die Klasse `SurfaceView` erweitert. Der `BarcodeManager` führt alle Manipulationen mit der Kamera durch und wird auf der Activity platziert. Wenn der Benutzer die Applikation startet, wird zu Beginn automatisch ein `AutoFocus`-Prozess erzeugt. Dieser wird so lange ausgeführt, bis ein Bereich innerhalb der Oberfläche „scharf“ gestellt werden kann. Danach wird automatisch ein Vorschaubild der Kamera fotografiert und an die Klasse `ManagePreviewImage` gesendet. In diesem externen Thread wird der C-Decoder aufgerufen, der das Foto analysiert und verarbeitet. Der Thread gibt das Resultat des Scannens an die Activity `ActCamera` weiter, die wiederum die Browser-Aktivität (`ActBrowser`) startet und dem Benutzer die Produkteinformationen anzeigt.

Falls der Decoder keinen Code ausfindig machen konnte, wird ein weiteres Vorschaubild ausgefiltert und an den Decoder gesendet. Bei negativem Resultat wird dieser Vorgang zweimal durchlaufen. Falls immer noch kein Code gefunden wurde, startet ein neuer `AutoFocus`-Prozess. Sobald die Kamera den `AutoFocus` erfolgreich abgesetzt hat, wird wiederum ein Vorschaubild an den Decoder gesendet. Dieser Prozess wiederholt sich so lange bis ein Code gefunden wurde, oder der Benutzer den „Zurück“-Button geklickt hat.

Wie bereits erwähnt, ist ein Flussdiagramm, dass das Prinzip des Scannens verdeutlicht ist im Anhang beigelegt.



## Die Activity / Das Layout

Das eigentliche Layout wird in der Klasse `BarcodeManager` zusammengestellt und in die `Activity` integriert. Damit die Kamera den vollen Bildschirm nutzen kann, muss zuerst die Titel-Leiste versteckt und dann die Flags `FLAG_FULLSCREEN` gesetzt werden.

Code-Abschnitt 6-6: Zum Fullscreen Modus wechseln

```
1 // Hide the title bar
2 requestWindowFeature(Window.FEATURE_NO_TITLE);
3
4 // Go into fullscreen mode
5 getWindow().setFlags(
6     WindowManager.LayoutParams.FLAG_FULLSCREEN,
7     WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

Aus `ActCamera.java`, Methode `onCreate()`

Die Funktion `requestWindowFeature()` und `getWindow()` wird in der Klasse `Activity` deklariert und kann somit von all seinen „Kinder“ aufgerufen werden.

Im nächsten Schritt wird eine Klasse instanziiert, die die Oberfläche der Kamera-Sicht abändert.

Damit nicht der ganze Bereich des Bildschirms analysiert werden muss, wird dieser begrenzt. Dies aus einfachem Grund: Je grösser das Bild, desto länger ist die Verarbeitungszeit der Decodierungs-Methode. Standardmässig wird der zu scannende Bereich auf 250\*150 Pixel beschränkt. Diese Werte sind in den Instanzvariablen `RECTANGLE_WIDTH` und `RECTANGLE_HEIGHT` gespeichert und können dementsprechend abgeändert werden.

So dass der Benutzer weiss, in welchem Rechteck er seinen Code platzieren soll, wird die Klasse `CanvasDrawer.java` instanziiert. Durch die Übergabewerte wird der gesamte Screen, sowie die Breite und Länge des Rechtecks definiert. Der `CanvasDrawer` zeichnet auf jeder Seite ein Quadrat und ändert seinen Alphawert. So erscheint in der Mitte der Oberfläche ein Rechteck, welches für den Scan fokussiert werden muss.

Abbildung 13: Veranschaulichung Kamera-Oberfläche



Quelle: Stefan Theler

Dadurch weiss der Benutzer an welchem Ort er seinen Barcode platzieren muss. Zudem erstellt der CanvasDrawer einen sprachabhängigen Text.

Code-Abschnitt 6-7: Erstellen der Rechtecke

```
1 Rect topRectangle = new Rect(  
2     0,  
3     0,  
4     myScreen.x,  
5     (myScreen.y / 2) - (rectY / 2));  
6 Rect bottomRectangle = new Rect(  
7     0,  
8     (myScreen.y / 2) + (rectY / 2),  
9     myScreen.x,  
10    myScreen.y);  
11 Rect rightRectangle = new Rect(  
12    (myScreen.x / 2 + rectX / 2),  
13    (myScreen.y / 2) - (rectY / 2),  
14    myScreen.x,  
15    (myScreen.y / 2) + (rectY / 2));  
16 Rect leftRectangle = new Rect(  
17    0,  
18    (myScreen.y / 2) - (rectY / 2),  
19    (myScreen.x / 2) - (rectX / 2),  
20    (myScreen.y / 2) + (rectY / 2));
```

Aus DrawCanvas.java, Methode onDraw()

Die Zeilen 1-5, 6-10, 11-15 und 16-20 zeigen die Definition der verschiedenen Rechte dar. Die Dimensionen werden jeweils aus der Grösse des gesamten

Screens, sowie der Grösse des gewünschten Rechtecks erstellt. Das Grössenverhältnis entspricht dem oben dargestellten Bild.

Allgemein wird ein Rechteck wie folgt definiert:

```
1 Rect myRect = new Rect(int bottom, int left, int right, int top);
```

Definition eines Rechtecks in Java Android

Nach der Definition der verschiedenen Rechtecke müssen diese auf einen Layer (Klasse `Paint`) gesetzt werden. Die verschiedenen `Paint`-Objekte werden am Ende der Manipulationen auf die Oberfläche (`Canvas`) plazierte.

```
1 Paint paintBorder = new Paint();
2 paintBorder.setStyle(Paint.Style.FILL);
3 paintBorder.setColor(Color.WHITE);
4 paintBorder.setAlpha(borderAlpha);
```

Code-Abschnitt 6-8: Beispiel eines Paint Objektes

## Die Kamera-Funktionalitäten

Als Basis für das Kamera-Handling wurde das freie Beispiel „CameraCanvas.java“ von Google genommen, von Grund auf überarbeitet und in der Klasse `BarcodeManager.java` integriert. Der Barcode Manager muss am Schluss nur in die Activity (`ActCamera.java`) eingebunden werden.

Das Ziel ist es, dem Benutzer die Bedienung so einfach wie möglich zu gestalten. Deshalb wurde die Applikation so programmiert, dass die Fotos der Barcodes ohne Druck auf den Auslöser aufgenommen werden. Android bietet hierfür eine effiziente Lösung. Das Abbild, welches auf den Bildschirm projiziert wird, stellt Android als Vorschaubild (Preview) zur Verfügung. Dies kann durch Aufrufen eines Preview-Callback-Handlers für die weitere Verarbeitung entnommen werden.

Sobald der Nutzer den Menüpunkt „Scannen eins Barcodes“ wählt wird automatisch ein Autofocus ausgeführt. Dies wird mit einer Callback-Funktion ausgeführt. Ein solche Rückruffunktion übergibt eine Methode als Parameter mit:

Code-Abschnitt 6-9: AutoFocus-Callback Funktion

```
1 private final Camera.AutoFocusCallback autoFocusCallback =  
2     new Camera.AutoFocusCallback() {  
3         public void onAutoFocus(boolean success, Camera camera) {  
4             if(isPreviewing) {  
5                 if (success) {  
6                     mCamera.setPreviewCallback(previewCallback);  
7                 } else {  
8                     mCamera.autoFocus(autoFocusCallback);  
9                 }  
10            }  
11        }  
12    };
```

Aus BarcodeManager.java

Die Variable `autoFocusCallback` wird generiert, um sie später der Methode `mCamera.autoFocus()` mitzugeben. Die Zeilen 3-10 definieren den `onAutoFocus`-Event. Dieser beantwortet die Frage: „Was passiert nach einem erfolgreichen Autofokus?“. Zuerst wird überprüft ob die Kamera überhaupt noch aktiv ist und danach ob der AutoFocus erfolgreich war. Falls dies der Fall ist, so wird die nächste Callback-Funktion aufgerufen. Falls der Autofokus nicht erfolgreich war, d.h. die Kamera hat keine Anhaltspunkte zur Scharfstellung gefunden, wird dieser wiederholt. Dies geschieht so lange bis ein Autofokus erfolgreich durchgeführt werden konnte oder der Benutzer den Vorgang abbricht.

Die Methoden `surfaceCreated()`, `surfaceDestroyed()` und `surfaceChanged()` der Klasse `BarcodeManager` sind für die Anzeige der Kamera verantwortlich. Sie wurden vom Google Beispiel übernommen und nur leicht angepasst. Damit die Kamera geöffnet wird, ruft man in `surfaceCreated()` die Methode `open()` des Kamera-Objekts auf und um sie wieder zu schliessen die Methoden `stopPreview()` und `release()`. Speziell zu erwähnen ist, dass bei der Methode `surfaceChanged()`, welche bei einem Neustart der Kamera-Activity aufgerufen wird, immer zuerst einen AutoFocus durchführt. So schliesst sich der Kreis und die Suche nach einem Barcode wird nicht unterbrochen.

Der `previewCallback`-Funktion implementiert die Methode `onPreviewFrame()`.

Code-Abschnitt 6-10: PreviewFrame-Callback Funktion

```
1 private final Camera.PreviewCallback previewCallback =
2     new Camera.PreviewCallback() {
3         public void onPreviewFrame(byte[] data, Camera camera) {
4             if(isPreviewing && mPictureThreadHandler != null) {
5                 try {
6                     camera.setPreviewCallback(null);
7                     if(data != null) {
8                         Message msg =
9                             mPictureThreadHandler.obtainMessage(
10                                0,
11                                mScreen.x,
12                                mScreen.y,
13                                data);
14                         msg.sendToTarget();
15                     }
16                 } catch(NullPointerException exception) {
17                     exception.printStackTrace();
18                 }
19             }
20         }
21     };
```

Aus BarcodeManager.java

Diese nimmt den aktuellen Bildschirm und „exportiert“ ihn als `byte`-Array. Nach diversen Tests (Ist der Thread gestartet? Sind Daten vorhanden?), versendet die Methode eine Nachricht (`Message`) an den externen Thread (Zeile 9-14). Der Thread ist für die Verarbeitung der verschiedenen Bilder verantwortlich und erhält folgende Daten innerhalb der Nachricht: Der erste Parameter gibt an, um welche Nachricht es sich handelt. In dieser Situation reicht es, wenn eine 0 angegeben wird, da immer der gleiche Nachrichtentyp versendet wird. Die weiteren Parameter sind optional und können frei gewählt werden. In diesem Beispiel werden die Länge und Breite des Bildschirms, sowie die Daten mitgeschickt.

## Die Datenverarbeitung

Damit die Daten des Bildes sauber verarbeitet werden können, sollte dies in einem externen Thread geschehen. Eine Klasse welche von `Thread` erbt, muss jeweils eine Methode `run()` implementieren. Diese Methode wird ausgeführt, so bald der Thread startet.

In diesem Fall ist der Thread für die Bildverarbeitung verantwortlich und wird ausgeführt, sobald er eine Nachricht (`Message`) von der Kamera-Activity erhält. Diese Nachricht wird mit einem Handler abgefangen und weiter verarbeitet. Da die Daten in einem `byte`-Array übermittelt werden und diese Form (YUV 420 - Format) nicht vom Decoder gelesen werden kann, müssen die Daten konvertiert

werden. Zuerst wird die Methode `convertIntoBitmap()` verwendet um die Daten in ein Bitmap zu wandeln.

Code-Abschnitt 6-11: Yuv420sp – Byte Array in ein Bitmap wandeln

```
1 public Bitmap convertIntoBitmap(byte[] yuv420sp,  
2     int screenWidth, int screenHeight) {  
3  
4     int width = frame.width();  
5     int height = frame.height();  
6     int[] pixels = new int[width * height];  
7  
8     for (int y = 0; y < height; y++) {  
9         int base = (y + frame.top) * screenWidth + frame.left;  
10        for (int x = 0; x < width; x++) {  
11            int grey = yuv420sp[base + x] & 0xff;  
12            pixels[y * width + x] = (0xff << 24) |  
13                (grey << 16) |  
14                (grey << 8) |  
15                grey;  
16        }  
17    }  
18    Bitmap bitmap = Bitmap.createBitmap(width, height,  
19        Bitmap.Config.ARGB 8888);  
20    bitmap.setPixels(pixels, 0, width, 0, 0, width, height);  
21    return bitmap;  
22 }
```

Aus ManagePreviewImageThread.java, Methode `convertIntoBitmap()`

Als Vorlage galt hierbei eine Methode aus einer anderen Open Source Android (ZXing<sup>1</sup>). Das byte-Array wird Pixel für Pixel in ein `int`-Array umgewandelt (Zeile 8 – 17) und am Schluss als Bitmap gespeichert und zurückgegeben (Zeile 18 – 21). Zuerst wird ein leeres Bitmap erzeugt (Zeile 18-19), welches danach mit den einzelnen Pixel aufgefüllt wird (Zeile 20). Dazu wird die Methode `setPixels(pixels, offset, stride, x, y, width, height)` verwendet. Der erste Parameter stellt die Daten dar, der zweite den Offset, also bei welchem Pixel das Bild beginnen soll (standardmässig 0). Der dritte Parameter zeigt die Anzahl der Farben, welche zwischen den Zeilen übersprungen werden sollen. Normalerweise wählt man hier die Breite des Bildes (`width`). Die Anfangs-Positionen und die Abmessungen des Bildes werden als letzte Übergabeparameter angegeben.

---

<sup>1</sup> <http://code.google.com/p/zxing/>

Nachdem die Daten im Bitmap Objekt gespeichert sind, wird es decodiert.

Code-Abschnitt 6-12: Vor bereiten des Decodierens

```
1 private boolean decoding(Bitmap bitmap) {
2
3     System.loadLibrary("codeicare-second");
4
5     ByteBuffer mybyteBuf = ByteBuffer.allocate(
6         bitmap.getWidth()*bitmap.getHeight()*4);
7
8     bitmap.copyPixelsToBuffer(mybyteBuf);
9
10    byte[] myArray = mybyteBuf.array();
11
12    mCode = decode(bitmap.getWidth(),bitmap.getHeight(),myArray);
13
14    if(mCode.equals("ERROR_NaN")) {
15        return false;
16    }
17    return true;
18 }
```

Aus ManagePreviewImageThread.java, Methode decoding()

Wie bereits erwähnt, war Decodierung in reinem Java zu langsam und deshalb wurde dies mit Hilfe eines nativen C-Decoders implementiert. Im Kapitel 6.5 wird die Decodierung genauer beschrieben.

In der Zeile 3 wird die „*Shared Library*“ des C-Decoders geladen. Danach wird das übergebene Bitmap in einen `ByteBuffer` (Zeile 5 – 8) umgewandelt. Dieser Schritt ist notwendig, weil der C-Decoder kein Bitmap als Übergabeparameter verlangt sondern wiederum ein `Byte`.

Nun stellt sich natürlich die Frage, warum man nicht direkt die rohen Daten decodieren kann und der „Umweg“ über das Bitmap-Objekt nötig ist. Doch das Problem hierbei ist, dass der Decoder die Bilder im YUV 420 Format nicht lesen kann. So ist dieser Zwischenschritt nötig. In einer späteren Version sollte es möglich sein, ein `Byte`-Array im YUV 420 Format direkt in das vom Decoder verwendete Format „ARGB“ zu konvertieren.

## Der Response Manager

Damit dem Benutzer auch eine Antwort, also Informationen angezeigt bekommt wurde ein Response Manager implementiert. Diese Klasse wird von der Kamera Activity, sowie von der manuellen Eingabe eines Codes verwendet.

Damit der Response Manager überhaupt aufgerufen werden kann, müssen die decodierten Daten erst einmal aus dem Thread kopiert werden. Dazu wird wiederum ein Handler verwendet, der eine Nachricht (`Message`) an die Kamera Activity sendet.

Code-Abschnitt 6-13: Code an Kamera Activity senden

```
1 try {
2     if(success) {
3         // Send the result to the Camera Thread
4         message = Message.obtain(
5             mCameraActivity.getHandler(), 0, mCode);
6     } else {
7         message = Message.obtain(mCameraActivity.getHandler(), 1);
8     }
9
10    message.sendToTarget();
11 } catch (NullPointerException e) {
12    e.printStackTrace();
13 }
```

Aus `ManagePreviewImageThread.java`, Methode `decodeImage()`

Zuerst wird getestet ob das Bild erfolgreich decodiert werden konnte. Wurde ein Code gefunden so wird dieser an den anderen Thread, also an die Kamera-Activity gesendet (Zeilen 4 – 5). Falls kein Code gefunden wurde, sendet die Nachricht keinen Inhalt.

Der jeweils erste Übergabeparameter der Methode `obtain()` gibt den Handler an, der die Nachricht erhält. Beim zweiten Übergabeparameter wird entweder eine 0, falls der Code gefunden wurde oder eine 1, falls der Code nicht gefunden wurde, gesendet. Bei einem positiven Resultat wird ein weiterer Parameter, der gefundene Code, mitgegeben.



Code-Abschnitt 6-14: Handler zum Verarbeiten des Resultates

```
1 public void handleMessage(final Message msg) {
2     if (mBarcodeManager.getCamera() != null) {
3         try {
4             if (msg.what == 0) {
5                 code = msg.obj.toString();
6                 counter = 0;
7
8                 User usr = new User(ActCamera.this);
9                 if (!usr.isUser()) {
10                     userId = "0";
11                 } else {
12                     userId = usr.getId();
13                 }
14
15                 String body = getString(
16                     R.string.scan_dialog_connection_body).replace("{0}",
17                     code);
18
19                 Dialog dialog = getUserAlertDialog(body);
20                 dialog.show();
21
22             } else {
23                 counter++;
24                 if (counter < 2) {
25                     mBarcodeManager.getCamera().setPreviewCallback(
26                         mBarcodeManager.getPreviewCallback());
27                 } else {
28                     counter = 0;
29                     mBarcodeManager.getCamera().autoFocus(
30                         mBarcodeManager.getAutoFocusCallback());
31                 }
32             }
33         } catch (NullPointerException e) {
34             e.printStackTrace();
35         }
36     }
37 }
38 }
```

Aus ActCamera.java, Methode handleMessage()

Der Handler, der die Nachricht empfängt und verarbeitet testet zuerst, ob ein Code gefunden wurde oder nicht. Falls das Resultat negativ ist, so wird zuerst der Counter inkrementiert (Zeile 23). Ist der Counter bereits 2-mal inkrementiert worden, d.h. es wurde 2-mal kein Code gefunden, so wird ein neuer Autofokus durchgeführt. Ist der Counter < 2 (Zeile 24) wird noch einmal ein Foto herausgefiltert und durch die Decoder-Klasse geschickt. Dies ist ein guter Mix zwischen Effizienz und Effektivität. Bei einem fehlerhaften Versuch ist es immer noch möglich einen Code zu erhalten. Doch wenn danach immer noch kein Code gefunden wurde, besteht Grund zur Annahme, dass die Kamera falsch fokussiert ist und so den Bereich des Codes nicht „scharf“ gestellt hat. So wird zur Sicherheit ein weiterer Autofokus durchgeführt. Dieser Vorgang

wiederholt sich solange bis ein Code gefunden wurde oder der Benutzer den Vorgang abbricht.

Wurde ein Code ausfindig gemacht, so wird dieser zuerst in ein String konvertiert (Zeile 5), dann wird geprüft, ob der Nutzer bereits registriert ist. Die Identifikationsnummer des Benutzers wird ausgelesen und in eine Variable gespeichert (Zeile 12).

Um den Benutzer zu benachrichtigen, dass nun eine Internetverbindung hergestellt wird, erscheint ein Dialog Box mit der Nachricht in der entsprechenden Sprache: „*Der Code ist XXX. Wollen Sie eine Internetverbindung herstellen?*“. Bestätigt der Benutzer diese Meldung so öffnet sich der Browser und der Code wird mit dem Server des RFID-Centers verglichen.

Wie bereits erwähnt kann neben dem Scannen eines Barcodes durch die Kamera, auch ein Code manuell eingegeben werden. Die Activity der Manuellen Eingabe des Codes wird mit Hilfe einer `EditView` implementiert und sendet nach einem Klick auf den Bestätigungsbutton den Code an den Response Manager. Die weiteren Schritte sind bei beiden Möglichkeiten exakt identisch. Folglich wird das Aufrufen des Response Managers und seine Antwort nur einmal beschrieben.

Code-Abschnitt 6-15: Aufruf des ResponseManagers

```
1 ResponseManager rspMng = new ResponseManager(  
2     ActCamera.this, userId, code);  
3  
4 rspMng.setUpHTTPConnection();  
5  
6 try {rspMng.doPost();  
7     if (!rspMng.handleResponse()) {  
8         mBarcodeManager.getCamera().autoFocus(  
9             mBarcodeManager.getAutoFocusCallback());  
10    }  
11  
12 } catch (ClientProtocolException e) {  
13     e.printStackTrace();  
14 } catch (IOException e) {  
15     e.printStackTrace();  
16 }
```

Aus ActCamera.java, Methode `getUserAlertDialog()`

Der Response Manager übergibt den aktuellen Kontext und die `UserId` des Benutzers mit (Zeilen 1 -2). Um ein Internetverbindung herzustellen muss eine `DefaultHttpClient` – Connection erstellt werden (Zeile 4).

Später sollen ebenfalls eine HTTPS – Verbindung, sowie Basic Authentication unterstützt werden. Eine Möglichkeit dies zum Implementieren wurde bereits in einem öffentlichen Forum gefunden<sup>1</sup> und im Package

---

<sup>1</sup> <http://www.anddev.org/httpsurlconnection-t5777.html>

`ch.rfidcenter.codeicare.ssl` abgelegt. Der Autor dieser Arbeit kann für die Richtigkeit dieser Methoden nicht garantieren, weil diese Teile noch nicht getestet werden können, da der Server des RFID-Centers diese Möglichkeiten noch nicht unterstützt. Deshalb kann in dieser Dokumentation nicht weiter auf die weiteren Verbindungsmöglichkeiten eingegangen werden. Das RFID-Center wird die HTTPS-Connection und Basic Authentication zu einem späteren Zeitpunkt selber weiter entwickeln.

Da die Daten mittels POST-Anfrage an den Server übermittelt werden, müssen die Variablen zunächst definiert werden.

Code-Abschnitt 6-16: Generierung der POST-Variablen

```
private void setUpPostVariables() {  
    postVariables = new ArrayList <NameValuePair>();  
    postVariables.add(new BasicNameValuePair("code", barcode));  
    postVariables.add(new BasicNameValuePair("version", getVersion()));  
    postVariables.add(new BasicNameValuePair("shortname", "codeicare"));  
}
```

Aus ResponseManager.java, Methode setUpPostVariables ()

Diese POST-Variablen werden mit Hilfe von Werte Paaren generiert und können so vom Server gelesen werden. Der Server benötigt drei POST-Variablen:

Tabelle 4: Postvariablen

code	Der Code den die Applikation aus dem Bild gelesen hat als String.
version	Die Version der Applikation. Diese wird mit Hilfe einer entsprechenden Methode und dem <code>PackageManager</code> ausgelesen.
shortname	Der Shortname ist der Kurzname der Applikation. Dieser ist bei jeder POST-Anfrage derselbe und zwar <i>codeicare</i> .

Nachdem die Postvariablen generiert und die Verbindung (HTTP) gewählt worden ist wird die POST-Anfrage ausgeführt. Die Methode `doPost()` übernimmt diesen Schritt:

Code-Abschnitt 6-17: POST Request durchführen

```
1 public void doPost() throws ClientProtocolException, IOException {  
2     httpPost.setEntity(  
3         new UrlEncodedFormEntity(postVariables, HTTP.UTF_8));  
4  
5     response = httpClient.execute(httpPost);  
6 }
```

Aus ResponseManager.java, Methode doPost ()

Die POST-Variablen werden dem Server übermittelt (Zeilen 2 – 3) und das Resultat in eine Variable des Types `HTTPResponse` gespeichert (Zeile 5).

Damit das Resultat analysiert und ausgelesen werden kann muss es in einen String konvertiert werden. Dies erledigt die Methode `getResponseString()`, welche die Entity des Resultats ausliest und konvertiert:

Resultat in einen String konvertieren.

```
1 EntityUtils.toString(response.getEntity());
```

Nachdem das Resultat erhalten und konvertiert worden ist, muss man dies noch analysieren und je nach Situation die entsprechenden Events starten. Folgende Möglichkeiten kann die Antwort des Servers aufweisen:

**Tabelle 5: Resultatmöglichkeiten des Servers als String**

CFG:<id>	Den User registrieren und eine entsprechende Meldung anzeigen.
CFG:<id>;<url>	Den User registrieren und ihn direkt auf eine Webseite umleiten.
<url>	Ein Produkt wurde gefunden und der Benutzer wird auf eine Webseite geleitet.
<ERROR>	Der Code konnte nicht identifiziert werden, der Service ist zurzeit nicht verfügbar oder ein anderer Fehler trat auf. Der Benutzer wird entsprechend benachrichtigt. Der Rückgabewert ist nicht <Error>, sondern der Typ der Fehlermeldung als String. Bsp: GEPiR_ERROR_TIMEOUT
<SERVICE_ADDED>	Dem Benutzer wurde ein neuer Service hinzugefügt und freigeschaltet.

Da die Antwort als String abgespeichert wurde, muss dieser in seine Bestandteile zerlegt werden und so auf die oben erwähnten Typen getestet werden:

Zunächst wird der String aufgesplittet und geschaut ob die ersten 4 Zeichen aus „CFG:“ bestehen. Falls ja wird der Benutzer registriert und falls nötig auf die neue Webseite umgeleitet. Beginnt der String nicht mit einem CFG: so muss mittels If-Statements getestet werden ob es sich um eine Fehlermeldung handelt. Folgende Fehlermeldungen sind möglich:

- GEPiR\_ERROR\_TIMEOUT
- GEPiR\_CONNECTION\_UNSUPPORTED\_ERROR
- GEPiR\_NO\_DATA\_FOUND
- NO\_DATA
- NO\_SERVICES

Die entsprechende Fehlermeldung wird mit Hilfe des String-Files eingelesen und dem Benutzer angezeigt. Handelt es sich bei der Antwort weder um eine Registrierung, einen neuen Service oder eine Fehlermeldung, so beinhaltet sie zwingend eine URL. D.h. alles ist wunschgemäss verlaufen, der Code konnte identifiziert werden und die Applikation ist bereit, den Benutzer an die Webseite weiterzuleiten.

Code-Abschnitt 6-18: Öffnen des Produktes

```
1 Intent newActivity = new Intent(mContext, ActBrowser.class);
2 newActivity.putExtra("url", uri);
3 mContext.startActivity(newActivity);
```

Aus ResponseManager.java, Methode showProduct()

Eigentlich könnte man die normale bereits integrierte Browser-Activity verwenden. Doch in diesem Fall hat man sich entschlossen eine eigene *WebView* zu verwenden, welche nur die Navigationsmöglichkeiten „Back“ und „Close“ bietet. Damit die neue Activity weiss, welche URL geöffnet werden muss, wird diese als Parameter mitgegeben (Zeile 2). Der erste Parameter der Methode `putExtras()` bestimmt der Name der Variable, der zweite den Inhalt.

## Die Web-View-Anzeige

Damit das Produkt angezeigt werden kann, wurde eine *WebView-Activity* namens `ActBrowser` entwickelt. Dieser Browser implementiert neben der Ansicht des Produktes zwei Buttons. Für diese wurden normale *OnClickListener* verwendet sie auch in Java SE ihre Verwendung haben.

Damit der Browser weiss, welche URL er zu öffnen hat, wurde ihm ein Übergabeparameter mitgegeben. Dieser wird folgendermassen wieder ausgelesen:

Lesen eines Übergabeparameters bei Activities

```
1 Bundle extras = getIntent().getExtras();
2 String myUrl = extras.getString("myParameter");
```

Aus ActBrowser.java, Methode onCreate()

Der Parameter kann danach wie normale Strings verwendet werden. In der CodeCarc Applikation wird dieses Vorgehen wie erwähnt bei der Übermittlung einer URL verwendet. Die Webseite wird mit Hilfe dem Kommando `WebView.loadUrl(extras.getString("url"))`; eingelesen und angezeigt.

## 6.5 Der native C-Decoder

### Was ist JNI

JNI ist das Java Native Interface und ermöglicht einem Entwickler das Aufrufen einer „*Shared Library*“ oder einer *Windows-DLL* unter Java. Um eine native Funktion aufzurufen, muss diese in der gewünschten Java-Klasse deklariert werden:

Einbindung einer native Methode

```
1 public native String decode(int Width, int Height, byte[] Data);
```

Der Ausdruck „native“ muss zwingend bei allen nativen Methoden implementiert werden.

## Die JNI Problematik

Zu Beginn dieser Bachelorarbeit wurde der Einbau von nativen Funktionen in Android Applikation von offizieller Seite nicht erlaubt. Zwar gab es bereits einige Entwickler, welche sich dieser Problematik stellten, doch eine einheitliche Lösung war nicht vorhanden.

Am 25. Juni 2009 veröffentlichte Google<sup>1</sup> eine erste Version des Native Development Kits (NDK). Mit Hilfe dieses NDK soll es den Entwickler ermöglicht werden, auf einfachste Weise nativen Code in die Applikation zu integrieren. Zu beachten gilt es, dass das NDK nur bei Android Projekten mit Versionscode 1.5 oder höher funktioniert.

Das Android NDK stellt folgende Funktionalitäten zur Verfügung<sup>2</sup>:

- Eine Auswahl von Tools und Build-Files um native Code-Bibliotheken von C und C++ zu erstellen.
- Ein Weg um diese nativen Bibliotheken in das bestehende Projekt einzubinden
- Eine Auswahl von nativen System Headers und Bibliotheken, welche in allen neuen Android Version (ab 1.5) verfügbar sind.
- Eine vollständige Dokumentation, sowie Beispiele und Tutorials

Das Ziel des NDK ist nicht, dass fortan Applikationen in C++ geschrieben werden. Sie soll Java Entwickler unterstützen, bereits geschriebenen C-Code in Java Applikationen zu integrieren. Um das Kit zu nutzen, ist ein gutes Verständnis von JNI und seinem Prinzip sehr wichtig. Eine gute Einführung in JNI kann hier abgerufen werden: <http://java.sun.com/docs/books/jni/html/intro.html> (Englisch, zuletzt abgerufen am 23. Juli 2009).

## Die Installation und Konfiguration der NDK<sup>3</sup>

In der Datei `/DOCS/INSTALL.TXT` sind alle Schritte zur Installation auf Englisch genau erklärt.

Zuerst muss das NDK vom Server der Android Projekt Site heruntergeladen werden: <http://developer.android.com/sdk>. Die aktuellste Version befindet sich unter dem Abschnitt *Native Development Tools (Zurzeit 1.5, R1 – Stand 23. Juli 2009)*.

Nach dem Download ist das NDK an einen geeigneten Ort (z.B: C:\Program Files\Android\NDK) zu platzieren. Die Ordnerstruktur der NDK sieht folgendermassen aus:

---

<sup>1</sup> <http://android-developers.blogspot.com/2009/06/introducing-android-15-ndk-release-1.html>

<sup>2</sup> Quelle : Android NDK Dokumentation 1.5

<sup>3</sup> Quelle : Dokumentation NDK 1.5

Tabelle 6: Ordnerstruktur NDK

/apps	Hier werden alle Applikationen, welche die NDK nutzen plaziert.
/build	Die Build-Files, welche zur Erstellung der C-Bibliotheken verwendet werden.
/docs	In /docs ist die gesamte Dokumentation des NDK gespeichert.
/out	Die Files, welche bei der Kompilierung und dem Build der Source-Files erstellt werden.
/sources	Die C / C++ Source Files nach Projekten geordnet

Falls das Android Projekt mit einer Windows Maschine entwickelt wird, muss man vorgängig den C-Compiler *cygwin* herunterladen und installieren. Alle nötigen Informationen zu *cygwin* erhält man auf der Webseite <http://www.cygwin.com/>. Es ist zu beachten, dass die Komponenten *make* (GNU Make 3.81 oder höher) und *gcc* vollständig installiert sein müssen. Bei Linuxdistributionen sind ist der C-Compiler meistens bereits vorhanden.

Es sind keine weiteren Installationsschritte notwendig. Das NDK sollte nach dem Herunterladen bereits funktionieren. Um die Konfiguration abzuschliessen und zu testen ist aus dem NDK Root Verzeichnis folgender Befehl (in der Command Line) auszuführen:

```
NDK Konfiguration abschliessen und testen  
C:\Program Files\Android\NDK> build/host-setup.sh
```

Falls keine Fehler auftreten wird das Configfile in `/out/host/config.mk` geschrieben. Hier befindet sich Informationen zur lokalen Maschine und welche Compiler verwendet werden.

## Die Verwendung des Android NDK

Nach dem Ausführen des Befehls `build/host-setup.sh` ist die Konfiguration abgeschlossen. Als Erstes müssen die C / C++-Source-Files in das Verzeichnis `<NDK DIR>/sources/<project_folder>/` plaziert werden.

Einen Teil der Sources für die CodeIcare Applikation wurden vom Bereits bestehenden C-Decoder kopiert. Zum Teil mussten Sources neu geschrieben werden.

**Tabelle 7: Inhalt /sources/codeicare/**

Android.mk	Das Android Makefile, welches zur Kompilierung und Erstellung der Library verwendet wird.
DataCode.h	Das Header-File, welches den kompletten C-Decoder enthält. Diese Datei wurde komplett vom Institut Icare zur Verfügung gestellt.
decoder.cpp	Das C++ File, das indirekt auf den DataCode-Header zugreift und die Bilder decodiert. Rückgabewert der Funktion decode ist entweder der gefundene Code oder <code>ERROR_NaN</code> .
decoder.h	Definiert alle Klassen, welche in decoder.cpp implementiert werden.
second.cpp	Dieses File dient als Schnittstelle für Java-Integration und implementiert das Header File <code>jni.h</code> . Diese Funktion wird vom Java Teil aufgerufen
Second.h	Der zum <code>second.cpp</code> gehörende Header-Teil der Funktion. Ist nötig damit die <i>Shared Library</i> erstellt werden kann.

## Das File Android.mk erklärt

Code-Abschnitt 6-19 Erstellen der NDK-Libraries

```

1 LOCAL_PATH:= $(call my-dir)
2 APP_CPPFLAGS+= -I$(LOCAL_PATH)/../codeicare
3 # first lib, which will be built statically
4 #
5 include $(CLEAR_VARS)
6 LOCAL_MODULE := codeicare-decoder
7 LOCAL_SRC_FILES := decoder.cpp
8
9 include $(BUILD_STATIC_LIBRARY)
10
11 # second lib, which will depend on and include the first one
12 #
13 #
14 include $(CLEAR_VARS)
15 LOCAL_MODULE := codeicare-second
16 LOCAL_SRC_FILES := second.cpp
17
18 LOCAL_STATIC_LIBRARIES := codeicare-decoder
19
20 include $(BUILD_SHARED_LIBRARY)

```

Aus NDK/sources/codeicare/Android.mk

Das Android.mk File muss mit der Variable `LOCAL_PATH` beginnen (Zeile 1). Diese Variable gibt den Pfad an, in welchem die lokalen Source-Files plaziert



sind. Der Wert `my-dir` gibt den aktuellen Pfad zurück. In Zeile 5 werden die `LOCAL_XX` Variablen, mit Ausnahme von `LOCAL_PATH` geleert. Die erste Library wird in den Zeilen 6-9 eingefügt. Sie definieren das Modul und die Source-Files und binden dies dann zu einer statischen Bibliothek zusammen.

Die zweite Library, die dann die *Shared Library* darstellt, wird in den Zeilen 14-20 erstellt. In der Zeile 18 gibt man an, welche Statischen Bibliotheken für diese *Shared Library* verwendet werden. Bei der CodeIcare-Applikation ist das die oben erstellte Library `codeicare-decoder`. Die Bibliothek wird schliesslich in der Zeile 20 erzeugt, wo das `BUILD_SHARED_LIBRARY`-Script aufgerufen wird. Dies generiert schliesslich die `lib<Name>.so`-Datei, die in das Projekt eingebunden wird und alle nötigen Informationen zum Aufruf der nativen Methode beinhaltet.

## Der Decoder kurz erklärt

Das File `decoder.cpp` implementiert den Header `DataCode.h` und `decoder.h`. Es beinhaltet die Methode `char* decode(int Width, int Height, int DataSize, int Grayscale, char* Data)`.

Code-Abschnitt 6-20: C-Decodierung eines Barcodes (Teil 1)

```
1 char* decode(int Width, int Height, int DataSize,
2   int Grayscale, char* Data) {
3
4   byte* SrcBuffer;
5   int SrcPitch;
6
7   if (Grayscale == 1)
8   {
9       // No need to convert
10      SrcBuffer= (byte*)Data;
11      SrcPitch= DataSize / Height;
12  }
```

Aus `decoder.cpp`, methode `decode()`

Ist die Variable `Grayscale = 1`, dann ist das Bild bereits in 8-Bit Graustufen konvertiert. D.h. es muss nichts mehr konvertiert und die Daten können direkt gefüllt werden.

Code-Abschnitt 6-21: C-Decodierung eines Barcodes (Teil 2)

```
13     else if (Grayscale == 4)
14     {
15         // Convert to grayscale
16         SrcPitch= (Width + 3) & ~3;
17         SrcBuffer= new byte[Height * SrcPitch];
18         DataCode::CopyBGRComponent (
19             1,
20             Width,
21             Height,
22             (byte*)Data,
23             DataSize / Height,
24             SrcBuffer,
25             SrcPitch);
26     }
```

Aus aus decoder.cpp, methode decode()

Falls die Variable `Grayscale` den Wert 4 besitzt, ist ein 32-Bit-True Color-Bild und muss dementsprechend konvertiert werden. Dies wird mit der Methode `CopyBGRComponent` der Klasse `DataCode` gemacht.

Code-Abschnitt 6-22: C-Decodierung eines Barcodes (Teil 3)

```
27     else
28     {
29         // Convert to grayscale
30         SrcPitch= (Width + 3) & ~3;
31         SrcBuffer= new byte[Height * SrcPitch];
32         DataCode::CopyBGRComponent (
33             1,
34             Width,
35             Height,
36             (byte*)Data,
37             DataSize / Height,
38             SrcBuffer,
39             SrcPitch);
40     }
```

Aus aus decoder.cpp, methode decode()

Ist der Wert der Variable `Grayscale` weder 1 noch 4, so wird das Bild ebenfalls in Graustufen konvertiert. Hierzu verwenden wir die Funktion `CopyBGRComponent` der Klasse `DataCode`. Im Gegensatz zu `CopyBGRComponent` konvertiert `CopyBGRComponent` ein 24-Bit True Color Bild.

Code-Abschnitt 6-23: C-Decodierung eines Barcodes (Teil 4)

```
44  int Status= DC.Read(Width, Height, SrcBuffer, SrcPitch, Text2);
45  int i= 0;
46
47
48  if (Status == Success ) {
49      // Copy the string
50      for (; Text2[i] != 0; i++)
51          {
52              Content[i]= Text2[i];
53          }
54
55  }
```

Aus aus decoder.cpp, methode decode()

Die Methode `Read()` der Klasse `DataCode` untersucht das, mittlerweile in Graustufen konvertierte, Bild auf Barcodes oder Datenmatrizen. Da die Methode `Read()` komplett vom Institut Icare übernommen wurde, wird diese an dieser Stelle nicht weiter beschrieben.

Nach der Decodierung des Bildes gibt die Funktion entweder `true` oder `false` zurück. Falls sich beim eingescannten Bild um einen Barcode handelt, wird die ID zurückgegeben und in `Content` abgespeichert.

Code-Abschnitt 6-24: C-Decodierung eines Barcodes (Teil 5)

```
57  // End the string
58  Content[i]= L'\0';
59
60  // Release
61  if (Grayscale != 1) {
62      delete [] SrcBuffer;
63  }
64
65  if(Status == Success )
66      return Content;
67  else
68      return "ERROR NaN";
69 }
```

Aus aus decoder.cpp, methode decode()

Am Schluss wird dem `char`-Array `Content` der Suffix `L'\0'` hinzugefügt und bei positivem Fund wird der Code zurückgegeben. Falls kein Code gefunden wird, so gibt die Methode `"ERROR_NaN"` zurück.

## Die Shared Library kurz erklärt

Code-Abschnitt 6-25: Native Decodier-Methode

```
1 jstring
2 Java_ch_rfidcenter_codeicare_core_ManagePreviewImageThread_decode(
3     JNIEnv * env, jobject thiz, jint Width,
4     jint Height , jbyteArray inArray ) {
5
6     char* data = new char[Width*Height*4];
7     env->GetByteArrayRegion (inArray,
8                             (jint)0,
9                             (jint)Width*Height*4,
10                            (jbyte*)data);
11
12     char * code = decode(Width,
13                         Height,
14                         Width*Height*4,
15                         4, data);
16
17     jstring jstrBuf ;
18     jstrBuf = env->NewStringUTF(code);
19     env->ReleaseByteArrayElements(inArray, (signed char*) data, 0);
20
21     return jstrBuf;
22 }
```

Aus aus second.cpp

Diese Methode wird aus dem Java-Teil heraus aufgerufen. Sie gilt also als Schnittstelle zwischen Java- und C-Code. Der etwas komisch gewählte Methodenname

`Java_ch_rfidcenter_codeicare_core_ManagePreviewImageThread_decode()` ist zwingend so zu nennen. Der Präfix `Java_` ist bei allen nativen Methoden anzugeben, ansonsten kann der Code nicht kompiliert werden. Dem Präfix folgen das Package und die Klasse, in welcher die native Methode aufgerufen wird. In diesem Falle wird die Methode aus der Klasse `ch.rfidcenter.codeicare.core.ManagePreviewThread.java` aufgerufen. Native Methoden sind nur gültig wenn sie die Übergabeparameter `JNIEnv * env`, `jobject thiz` besitzen. Alle weiteren Parameter sind frei wählbar. Vor jeder Typenbezeichnung ist der Präfix „j“ zu plazieren (z.B. `jint`, `jstring`). Die weiteren Übergabeparameter bei dieser Funktion sind die Höhe, die Breite und das Bild als `ByteArray`.

Die Funktion `env->GetByteArrayRegion` (Zeile 7-10) kopiert eine Region eines Arrays in einen Buffer. Die Funktion `decode()` (Zeile 12-15) ruft `decode()` im oben beschriebenen File `decoder.cpp` auf. Der Rückgabewert stellt entweder der Code oder den String „`ERROR_NaN`“ dar. Der String muss nun noch Java-komform formatiert werden und wird danach zurückgegeben. Der Rückgabewert wird danach in der Klasse `ManagePreviewImageThread` weiter verwendet und verarbeitet.

## 6.6 Das Preferences Handling

### Allgemein

In Preferences können Konfigurationsdaten einer Applikation ausgelesen in eine Art Textfile abgespeichert werden. Die Preferences lassen sich vom Benutzer beliebig ändern und kann so die Applikation entscheidend beeinflussen.

Das Preference-Handling ist ein Teil der Android Guidelines und sollte möglichst immer als Konfigurations-Assistent verwendet werden. Android bietet von Haus aus verschiedenste Preference-Typen an. So z.b. gibt es eine Checkbox-Preferences oder auch List-Preferences. Bei Checkbox-Preferences kann die Option entweder ein oder ausgestellt werden. Bei der List-Preference muss ein Eintrag aus einer Liste gewählt werden. Es gibt eine ganze Liste<sup>1</sup> von Preferences, welche hier nicht im Detail angeschaut werden können.

### Die Sprachen

Die Applikation wird in 3 verschiedenen Sprachen angeboten. Englisch als Standardsprache, Französisch und Deutsch. Wie bereits in einem früheren Kapitel erwähnt, muss bei Mehrsprachigkeit nur neue Resource-Files erstellt werden (values-de, values-fr, ...). Standardmässig ist die Applikation in der Sprache eingestellt, in welcher das Betriebssystem läuft. Doch der Benutzer kann seine Sprache manuell wählen. Folgende Optionen stehen zur Verfügung:

- Automatisch
- Englisch
- Französisch
- Deutsch

Bei „*Automatisch*“ wird jeweils die Sprache des Handys verwendet, oder - falls nicht vorhanden - Englisch.

---

<sup>1</sup> <http://developer.android.com/reference/android/preference/package-summary.html>

Die Sprach-Präferenzen werden folgendermassen eingelesen:

Code-Abschnitt 6-26: Erstellung der Sprach-Präferenz

```
1 prefSetLanguage = new ListPreference(mContext);
2 prefSetLanguage.setEntries(R.array.conf_listprefs_language);
3 prefSetLanguage.setEntryValues(
4     R.array.conf_listprefs_language_value);
5 prefSetLanguage.setKey(KEY_LANGUAGE);
6 prefSetLanguage.setDefaultValue("automatic");
7 prefSetLanguage.setTitle(R.string.conf_lang_title);
8 prefSetLanguage.setDialogIcon(
9     android.R.drawable.ic_dialog_alert);
10 prefSetLanguage.setDialogTitle(
11     R.string.conf_lang_title);
12 prefSetLanguage.setSummary(
13     mContext.getString(
14         R.string.conf_lang_description) + " " + getCurrentLanguage());
```

Aus ConfigurationManager.java, Methode setPreferences()

Die Texte, sowie die Werte für die Auswahl der Sprache werden aus einem Array, welches im Ressourcen-File *arrays.xml* gespeichert ist, ausgelesen.

String-Array mit den verschiedenen Sprachen

```
<string-array name="conf_listprefs_language">
    <item>Automatic</item>
    <item>French</item>
    <item>English</item>
    <item>German</item>
</string-array>
```

Aus arrays.xml

Es ist wichtig die Werte (`setEntryValues()`) von den Anzeigetexten (`setEntries()`) zu unterscheiden, denn die Werte werden später beim auslesen der Einstellungen verwendet. Die Anzeigetexte werden nur für die GUI gebraucht und haben keinen Einfluss auf das eigentliche Programm. Deshalb wurden auch nur die Anzeigetexte in den jeweiligen Sprachen übersetzt.

Ausgelesen wird die Spracherkennung zu Beginn der Applikation, also beim Scannen eines Barcodes:

Code-Abschnitt 6-27: Einbindung der Sprachwahl

```
1 SharedPreferences sharedPref =
2     PreferenceManager.getDefaultSharedPreferences(this);
3 String language =
4     sharedPref.getString("set_language", "not_set");
5
6 Resources res = this.getResources();
7 DisplayMetrics dm = res.getDisplayMetrics();
8 Configuration conf = res.getConfiguration();
9
10 // Choose language
11 if(language.equals("german")) {
12     conf.locale = Locale.GERMAN;
13     res.updateConfiguration(conf, dm);
14 } else if(language.equals("french")) {
15     conf.locale = Locale.FRENCH;
16     res.updateConfiguration(conf, dm);
17 } else if(language.equals("english")) {
18     conf.locale = Locale.ENGLISH;
19     res.updateConfiguration(conf, dm);
20 } else {
21     conf.locale = Locale.getDefault();
22     res.updateConfiguration(conf, dm);
23 }
```

Aus ActCamera.xml, Methode chooseLanguage()

Zuerst werden alle Präferenzen ausgelesen (Zeilen 1 – 2) und in ein `SharedPreferences` Objekt gespeichert. In `SharedPreferences` können alle Präferenzen abgespeichert werden, die in der gesamten Applikation verfügbar sein müssen. Durch den `PreferenceManager` können diese ausgelesen werden.

Um nun eine spezifische Konfigurations-Einstellung zu erhalten, muss man deren Identifier (in diesem Falle „*set\_language*“) wissen und in der Methode `getString()` angeben. Das Resultat kann nun in ein String gespeichert (Zeilen 3 – 4) und für weitere Manipulationen verwendet werden.

Um eine neue Sprache zu wählen müssen die Locale-Einstellungen geändert und deren Konfiguration neu geladen werden. Dies wird in den Zeilen 11 – 22 gemacht. Die Applikation testet, welche Sprache gerade gewählt ist und aktualisiert diese wenn nötig.

## Die Verwendung der History

Durch die Checkbox-Präferenz wird definiert, ob der Verlauf genutzt wird oder nicht. Der Verlauf speichert alle bisher gescannten Produkte an und kann sie auf Wunsch wieder anzeigen.

Diese Funktionalität kann in den Preferences eingestellt werden. Will der Benutzer nicht, dass seine Werte gespeichert werden, so wählt er die Checkbox ab. Ab diesem Zeitpunkt werden die Scannresultate nicht mehr gespeichert.

Der Wert der History-Preference wird in der Klasse `ResponseManager` ausgelesen, wo getestet wird ob der Verlauf verwendet werden soll oder nicht.

Code-Abschnitt 6-28: Einbindung der History Preference

```
1 SharedPreferences sharedPref =  
2     PreferenceManager.getDefaultSharedPreferences(mContext);  
3  
4 boolean history =  
5     sharedPref.getBoolean("activate_history", false);  
6  
7 if(history) {  
8     Date now = new Date();  
9     HistoryManager hm = new HistoryManager(mContext);  
10  
11     hm.add(barcode, uri, now.getTime());  
12 }
```

Aus `ResponseManager.java`, Methode `showProduct()`

Wie bei der Sprachwahl werden alle Preferences ausgelesen aber nur die spezifische („*activate\_history*“) weiter verwendet. Da der Wert der Einstellung nicht wie bei der Spracheinstellung ein `String` ist, sondern ein `Boolean` wird der dieser mit der Methode `getBoolean()` ausgelesen (Zeilen 4 – 5). Nur falls der Wert `true` ergibt, wird der Verlaufseintrag gespeichert. Das History-Handling wird im Kapitel 6.7 genauer beschrieben.



## 6.7 Das History Handling

### Die Idee

Damit ein Benutzer den gleichen Code nicht mehrmals einscannen muss und möglicherweise auch später noch einmal Zugriff auf diesen erhalten will, wurde ein Verlauf implementiert. So hat der Benutzer die Möglichkeit so oft wie möglich auf ein bereits gescanntes Produkt zuzugreifen. Das Wiederholte Auslesen des Barcodes ist demnach überflüssig geworden.

### Die Technik

Abbildung 14: Klassenaufteilung des Verlaufs

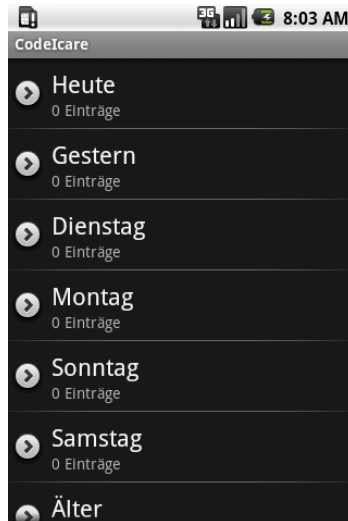


Quelle: Microsoft Visio Stefan Theler

Für den Verlauf wurden 3 verschiedene Klassen implementiert. Zum einen wird eine Aktivität verlangt, welche die Einträge anzeigt, des Weiteren ist ein Manager programmiert worden, der die Logik und die Anfragen der Aktivität verarbeitet. Um die Datenbank-Manipulationen durchzuführen, verwendete man einen `HistoryDbAdapter`. Der Adapter übernimmt die Anfragen des Managers und teilt diese an die Datenbank weiter.

## Die Anzeige des Verlaufs

Abbildung 15: Verlaufsanzeige



Quelle: Screenshot codeicare

Um die Anzeige des Verlaufs möglichst übersichtlich zu gestalten wurde eine `ExpandableList-View` verwendet, welche in Tagen eingegliedert ist. Klickt man auf einen Tag, so werden all Seine Einträge angezeigt. Aus diesem Grund wird auch immer das Datum mitgespeichert.

Standardmässig werden immer die letzten 6 Tage angezeigt. Alle älteren Einträge sind im Bereich „Älter“ gespeichert.

Da die Activity von der Klasse `ExpandableListActivity` erbt, kann zur Anzeige die Methoden `onOptionsItemSelected()` und `onChildClick()` implementiert werden. Diese Methoden sind verantwortlich dafür, dass bei einem Klick auf einen Tag, bzw. auf einen Eintrag ein Event gestartet wird. Bei einem Klick auf den Tag wird die Ansicht erweitert

und bei einem Klick auf einen „Child“-Eintrag der Browser mit dem entsprechenden Produkt angezeigt.

Folgende Codezeilen beziehen sich auf die Methode `generateHistory()` in der Klasse `ActCamera.java`.

Code-Abschnitt 6-29: Erstellen der History (Teil1)

```
1 // Create a new Instance of HistoryManager
2 mHistoryManager = new HistoryManager(this);
3
4 // Gets the language addicted weekdays from the string.xml file
5 mWeekdays = mHistoryManager.getWeekDays();
6
7 // Creates the parent and child lists
8 List<Map<String, String>> groupData =
9     new ArrayList<Map<String, String>>();
10 List<List<Map<String, String>>> childData =
11     new ArrayList<List<Map<String, String>>>();
12
13 GregorianCalendar today = new GregorianCalendar();
```

Aus `ActHistory.java`, Methode `generateHistory()`

Um den Verlauf zu genieren werden zuerst die aktuellen Wochentage ausgelesen (Zeile 5) und in die jeweilige Sprache übersetzt. Anschliessend muss eine Liste definiert werden, in der die jeweiligen Tage gespeichert sind (Zeilen 8 – 13).

Nach der Deklaration der Variablen, müssen diese nun gefüllt werden. Dazu wird eine `for`-Schleife verwendet, welche die „Durchleuchtung“ der letzten 6 Tage beinhaltet. Innerhalb dieser Schleife werden verschiedene Operationen durchgeführt:

Code-Abschnitt 6-30: Erstellen der History (Teil2)

```
1 //Generate the daily timestamp from the specific date
2 GregorianCalendar gc = new
3     GregorianCalendar(
4         today.get(GregorianCalendar.YEAR),
5         today.get(GregorianCalendar.MONTH),
6         today.get(GregorianCalendar.DAY_OF_MONTH));
7
8 // Creates the current Map entry
9 Map<String, String> curGroupMap =
10     new HashMap<String, String>();
11 // Add the current Map to the list
12 groupData.add(curGroupMap);
13
14 // Determines the day of week (0 = Sunday, 1 = Monday etc.)
15 int weekday =
16     today.get(GregorianCalendar.DAY_OF_WEEK);
17
18 // Gets the information from the database
19 Cursor cur =
20     mHistoryManager.getEntriesForADate(gc.getTimeInMillis());
```

Aus ActHistory.java, Methode generateHistory()

Zunächst wird eine Variable des Typs `GregorianCalendar` verwendet, der den aktuellen Tag ausliest (Zeilen 2 – 6). Nach diesem Schritt muss eine `Map` definiert werden, die den aktuellen Tag, sowie die einzelnen Einträge in sich speichert. (Zeile 8 – 11). In Zeile 12 wird der, zurzeit noch leere, Tag zur Liste hinzugefügt. Damit der Tag eindeutig definiert werden kann wird dieser als `Integer` ausgelesen und in der Variable `weekday` gespeichert (Zeilen 15 – 16).

Damit die Daten aus der Datenbank gelesen werden können, wird ein `Cursor` verwendet. Die Klasse `HistoryMangaer`, die der Datenbankklasse die Statements übergibt wird aufgefordert, alle Daten eines Datums herauszulesen (Zeile 20). Der Übergabeparameter ist der Timestamp. Für den Erhalt dieses Timestamps ist die Methode `GregorianCalendar.getTimeInMillis()` zuständig (Zeile 20).

Code-Abschnitt 6-31: Erstellen der History (Teil3)

```
1 String name;
2 switch(i) {
3     case 0:
4         name = getString(R.string.history_today);
5         break;
6     case 1:
7         name = getString(R.string.history_yesterday);
8         break;
9     case 6:
10         cur = mHistoryManager.getOlderEntriesAs(
```

```
11     today.getTimeInMillis());
12     name = getString(R.string.history_older);
13     break;
14     default:
15         name = mWeekdays[weekday-1];
16     break;
17 }
```

Aus ActHistory.java, Methode generateHistory()

Um die Anzeige benutzerfreundlicher zu gestalten wurden für die Namen folgendes Schema verwendet: Der Titel für den heutigen und gestrigen Tag wird mit „*Heute*“ und „*Gestern*“ angezeigt. Die anderen Einträge mit den jeweiligen Wochentagen. Dies ist für einen Benutzer besser zu lesen und verstehen als Datumsangaben. Deshalb muss hier ein `switch`-Statement die Tage auslesen und die jeweiligen Wochentage setzen. Alle Angaben werden aus dem sprachabhängigen String-Ressourcen-File ausgelesen.

Damit auch die älteren Beiträge angezeigt werden können, wird der Cursor `cur` noch einmal überschrieben (Zeilen 10 – 12). Eine entsprechende Methode in der Klasse `HistoryManager` liest alle Einträge aus, die älter sind als der angegebene Tag.

Code-Abschnitt 6-32: Erstellen der History (Teil4)

```
1     curGroupMap.put(NAME, name);
2
3     // Creates the children List
4     List<Map<String, String>> children =
5         new ArrayList<Map<String, String>>();
6
7     String label;
8
9     // Gets the number of history entries for a specific day
10    if(cur.getCount() == 1) {
11        label = getString(R.string.history_entry);
12    } else {
13        label = getString(R.string.history_entries);
14    }
15
16    curGroupMap.put(NUMENTRIES, cur.getCount() + " " + label);
17
18    int codeColumn = cur.getColumnIndex("code");
19    int urlColumn = cur.getColumnIndex("url");
20
21    // Go to the first entry
22    cur.moveToFirst();
23
24    ArrayList<String> temp2 = new ArrayList<String>();
```

Aus ActHistory.java, Methode generateHistory()

Nun wird der Name des Tages in die Map gespeichert (Zeile 1). Eine weitere Liste erzeugt die einzelnen Einträge für den entsprechenden Tag. Diese werden danach in die Liste `childData` eingefügt. Die Liste `children` besitzt ein Label

und eine URL. Das Label beinhaltet den Titel des Eintrages, also den Barcode und die URL die Webseite, auf welche der Eintrag verweist.

Damit der Benutzer weiss, wie viele Einträge zu den jeweiligen Tagen gehört, wird diese aus dem Cursor gelesen und in die Map des aktuellen Tages gespeichert (Zeile 16). Damit alle Verlaufseinträge ausgelesen werden können, ist es nötig, den Cursor an den Anfang zu plazieren (Zeile 22).

Code-Abschnitt 6-33: Erstellen der History (Teil5)

```
1 // Check if the cursor is at the beginning
2 if(cur.isFirst()) {
3
4     do {
5
6         // Get the columns
7         String code = cur.getString(codeColumn);
8         String url = cur.getString(urlColumn);
9
10        temp2.add(url);
11
12        // Add the columns to the Map
13        Map<String, String> curChildMap =
14            new HashMap<String, String>();
15        children.add(curChildMap);
16
17        curChildMap.put(CODE, mHistoryManager.matchString(code, 20));
18        curChildMap.put(URL, mHistoryManager.matchString(url, 35));
19
20    } while (cur.moveToNext());
21 }
22
23 temp.add(temp2);
24 // Adds the children to the list
25 childData.add(children);
26
27
28 // Decrement one day
29 today.add(GregorianCalendar.DATE, -1);
30 }
```

Aus ActHistory.java, Methode generateHistory()

Danach müssen die Einträge aus der Datenbank ausgelesen werden. Dazu wird der Cursor immer wieder neu gesetzt und verschoben. Die einzelnen Einträge werden in die Map gespeichert, wobei der Identifier `CODE` den Barcode und `URL` die dazugehörige Webseite bedeutet. Als Titel wird der Barcode angegeben und in der zweiten Zeile die URL (Zeilen 17 – 19).

Damit die verschiedenen Strings nicht abgebrochen werden, wird das Suffix „...“ bei allen Einträgen hinzugefügt, die eine gewisse Länge überschreiten. Dafür ist die Methode `matchString()` verantwortlich (Zeilen 17 – 18).

Nachdem die Einträge ausgelesen worden sind, wird der Tag dekrementiert, d.h. das gleiche wird mit dem gestrigen Tage wieder ausgeführt (Zeile 29).

Code-Abschnitt 6-34: Erstellen der History (Teil6)

```
1 // Creates the Simple Adapter
2 mAdapter = new SimpleExpandableListAdapter(
3     this,
4     groupData,
5     android.R.layout.simple_expandable_list_item_2,
6     new String[] { NAME, NUMENTRIES },
7     new int[] { android.R.id.text1, android.R.id.text2 },
8     childData,
9     android.R.layout.simple_expandable_list_item_2,
10    new String[] { CODE, URL },
11    new int[] { android.R.id.text1, android.R.id.text2 }
12 );
13 setListAdapter(mAdapter);
```

Aus ActHistory.java, Methode generateHistory()

Die einzelnen Einträge werden nach der `for`-Schleife mit Hilfe eines `SimpleExpandableListAdapter` auf die Seite projiziert. Die Parameter sind unterteilt in Daten und Layout. Zuerst gibt man die einzelnen Daten an, danach das gewünschte Layout. In diesem Fall wird das Standardlayout für eine Expandable-List verwendet. Des Weiteren müssen die einzelnen Id's angegeben werden, damit die Liste weiss wo und wann er die jeweiligen Einträge ausgeben soll (Zeilen 6 – 10). Am Schluss wird die Liste der Activity hinzugefügt (Zeile 13).

## Das Erstellen der Datenbank

Folgende Datenfelder wurden in der Datenbank definiert:

**Tabelle 8: Datenbanktabelle codeicare\_history**

<u>id</u>	Integer, PK, AUTO	Die ID der Zeile
url	Text, Not Null	Die URL des gescannten Produktes
code	Text, Not Null	Der Barcode des Produktes
tstamp	Text, Not Null	Das Datum und die Zeit, wann das Produkt eingescannt wurde.

Das Datum und die Zeit wird als Timestamp gespeichert. Ein Timestamp ist ein Zeitstempel, welcher die Sekunden seit dem 1. Januar 1970 00:00 zählt. Dieses Datum gilt als Unix Epoche und Beginn der Unix Zeitzählung. Aus einem Timestamp lässt sich sehr einfach ein Datum und eine Uhrzeit ablesen. Die Umrechnung wird von Standard-Java-Bibliotheken übernommen. So muss sich der Entwickler keine Sorgen um die Richtigkeit der Angaben machen.

Im Feld URL und Code werden die eigentlichen Daten abgespeichert, die in der Applikation angezeigt werden. In einer späteren Version könnte man anstatt des Barcodes der Produktetitel anzeigen lassen. Dies wäre noch aussagekräftiger.

Die Datenbankmanipulation des Verlaufs wurde alle in die Klasse `HistoryDbAdapter` ausgegliedert. So wird die Datenbank von der Applikations-

Logik bestmöglich getrennt. Als Basis für diese Klasse wurde das Notepad Tutorial<sup>1</sup> von Android verwendet.

Die Datenbank SQLite arbeitet, wie der Name schon sagt, mit SQL und versteht somit die einfachen Befehle wie `CREATE TABLE`, `SELECT` etc. Das Statement zum erstellen der Tabelle „*codeicare\_history*“ der Datenbank „*codeicare.db*“ lautet wie folgt:

Code-Abschnitt 6-35: SQL-Statement zum Erstellen der Tabelle

```
1 CREATE TABLE codeicare_history (  
2     id      INTEGER PRIMARY KEY AUTOINCREMENT,  
3     code    TEXT NOT NULL,  
4     url     TEXT NOT NULL,  
5     tstamp  TEXT NOT NULL)
```

Der Ausdruck `CREATE TABLE` (Zeile 1) erstellt die Tabelle, die zum Speichern der Datensätze verwendet wird. Die einzelnen Felder werden Zeile für Zeile mit den mit den verschiedenen Optionen eingefügt. Die Id z.B. erhält den Typ eines Integers, wird als Primärschlüssel verwendet und automatisch inkrementiert (Zeile 2). Ein Primärschlüssel ist eine eindeutige, nur einmal vorkommende Zeichenfolge, die den Datensatz eindeutig definiert. Datensatzmanipulationen können mit Hilfe eines Primärschlüssels durchgeführt werden.

Die anderen Datenfelder sind jeweils vom Typ `Text` und dürfen nicht `null` sein. D.h. bei jedem Datensatz muss ein Wert im jeweiligen Feld eingetragen sein.

## Das Einfügen eines History-Eintrages

Um einen neuen History-Eintrag einzufügen wird die Methode `add()` in der Klasse `HistoryDbAdapter` verwendet. Durch die Parameter welche vom Manager geschickt wurden kann ein Datensatz gefüllt werden.

Code-Abschnitt 6-36: Einfügen eines History-Eintrages

```
1 public long add(String code, String url, long date) {  
2  
3     ContentValues initialValues = new ContentValues();  
4     initialValues.put(KEY_CODE, code);  
5     initialValues.put(KEY_URL, url);  
6     initialValues.put(KEY_TS, date);  
7  
8     return mDb.insert(DATABASE_TABLE, null, initialValues);  
9 }
```

Aus `HistoryDbAdapter.java`, Methode `add()`

In den Zeilen 3 bis 6 werden die einzelnen Einträge generiert und in einem `ContentValues`-Objekt abgelegt. Diese Android-Klasse wird verwendet um Daten in eine Datenbank zu speichern. Der eigentliche Speichervorgang wird in

---

<sup>1</sup> <http://developer.android.com/guide/tutorials/notepad/index.html>

der Zeile 8 beschrieben. Durch die Methode `insert()` wird der neue Datensatz in der Datenbank erzeugt.

## Das Löschen eines History-Eintrages

Verlaufseinträge können nicht abgeändert, sondern nur gelöscht werden und zwar alle gemeinsam. Zurzeit ist es nicht erlaubt einzelne Einträge zu editieren oder zu löschen. Die Applikation unterstützt nur das Löschen von allen Einträgen gleichzeitig, also das Leeren der Tabelle.

Die Methode `clearTable()` ist für die Leerung der Tabelle verantwortlich:

Code-Abschnitt 6-37: SQL-Statement zum Erstellen der Tabelle

```
1 public int clearTable() {  
2     return mDb.delete(DATABASE_TABLE, "1", null);  
3 }
```

Der zweite Übergabeparameter gibt an, dass die gesamte Tabelle geleert werden soll. Im eigentlichen Sinne wird folgendes Statement an die Datenbank gesendet:

SQL-Statement zum Löschen der Tabelle

```
1 DELETE * FROM codeicare_history WHERE 1
```

## Das Auslesen eines History-Eintrages

Es existieren zwei verschiedene Arten, wie die Einträge gelesen werden. Zum einen werden alle Werte eines spezifischen Datums ausgelesen. Auf der anderen Seite können jedoch auch alle Einträge die älter sind als ein bestimmtes Datum ausgelesen werden. Um dies zu unterscheiden erhält die Methode `get()` einen zusätzlichen Parameter, welcher `type` genannt wird. Ist dieser Parameter = 0, so werden alle Einträge für ein spezifisches Datum herausgelesen. Bei 1 werden alle Einträge, die älter als das angegebene Datum sind, zurückgegeben.

```
1 public Cursor get(long tstamp, int type) {  
2     switch(type) {  
3         case 0:  
4             // Get the entries for a specific date  
5             return mDb.query(DATABASE_TABLE,  
6                 new String[] {  
7                     KEY_ROWID,  
8                     KEY_CODE,  
9                     KEY_URL,  
10                    KEY_TS},  
11                    KEY_TS + " < " + String.valueOf(tstamp),  
12                    null, null, null, null);  
13         case 1:  
14             // Get all older entries than a specific date  
15             long day1 = tstamp;  
16             long day2 = tstamp + 86400000;  
17             return mDb.query(DATABASE_TABLE,  
18                 new String[] {
```



```
19             KEY ROWID,
20             KEY_CODE,
21             KEY URL,
22             KEY_TS},
23             KEY_TS + " BETWEEN " +
24             String.valueOf(day1) + " AND " +
25             String.valueOf(day2),
26             null, null, null, null);
27     }
28     return null;
29 }
```

Aus HistoryDbAdapter.java, Methode get()

Bei beiden Arten ist der Vorgang eigentlich derselbe. Man schickt indirekt ein `SELECT` Statement an die Datenbank. Der Methode `query()`, werden alle nötigen Parameter, die für die Erstellung des Statements nötig sind, mitgegeben (Zeilen 5 – 12 und Zeilen 17 – 26). Der erste Parameter gibt die entsprechende Tabelle an, dann werden die gewünschten Felder definiert, die ausgelesen werden sollen. Als dritter Parameter weist auf die `WHERE` Klausel hin (Zeile 11 und Zeilen 23 – 25). Einzig hier unterscheiden sich die beiden Arten. Einmal wird nur ein Datum ausgelesen, beim anderen mehrere.

Wurden Einträge gefunden, werden diese mit einem `Cursor` zurückgegeben. Danach werden die Einträge in der Activity, wie oben beschrieben dargestellt.

## 6.8 Das Veröffentlichen der Applikation

### Die Voraussetzungen / Vorbereitungen

Um eine Applikation zu veröffentlichen muss man unbedingt folgende Voraussetzungen erfüllen:

- Die Applikation muss mit einem privaten Schlüssel signiert sein.
- Im `AndroidManifest.xml` muss der Versions-Code, sowie der Name der aktuellen Version eingegeben werden (`android:versionCode` und `android:versionName`)
- Im Manifest muss zudem ein Icon und eine „Inscription“ angegeben werden. (`android:icon` und `android:label`)

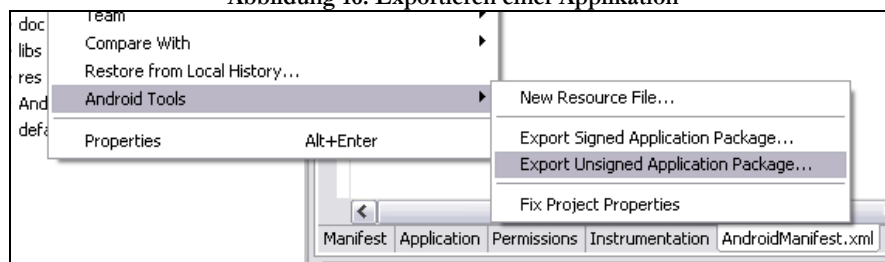
Teil des Android Manifest Files.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest
3     xmlns:android="http://schemas.android.com/apk/res/android"
4         package="ch.rfidcenter.codeicare"
5         android:versionCode="1"
6         android:versionName="1.0" >
7     <application
8         android:icon="@drawable/icon"
9         android:label="@string/app_name"
10        android:debuggable="true">
11        [ ... ]
12    </application>
13    <uses-sdk android:minSdkVersion="3" />
14    [ ... ]
15 </manifest>
```

## Das .apk-File erstellen

Alle Android Applikationen verwenden als Endung „.apk“. Diese kann mit Hilfe von Eclipse sehr einfach erstellt werden. Man wählt das Projekt aus und öffnet das Kontextmenü mit der rechten Maustaste. Praktisch am Ende der Liste befindet sich der Eintrag „Android Tools“. Wählt man diesen an, erscheint eine Liste, in der man den Eintrag „Export Unsigned Application Package...“ auswählt.

Abbildung 16: Exportieren einer Applikation



Quelle: Eclipse IDE

Nach dem Klick auf den Eintrag, muss der Entwickler einen geeigneten Platz für die Applikation wählen und diese dort abspeichern.

## Das digitale Signieren

Es ist zwingend nötig, dass eine für die Öffentlichkeit bestimmte Applikation digital signiert wird. Für andere Plattformen ist ein Kauf von teuren Zertifikaten unerlässlich, doch bei Android-Applikationen reicht es aus, wenn man einen eigenen privaten Schlüssel definiert und die Applikation mit diesem signiert.

Damit ein eigenes Signieren möglich ist, muss ein digitales Zertifikat verwendet werden. Dieses Zertifikat kann mit dem in der Java JDK integrierten Tool `keytool.exe` manuell erstellt werden.

Befehl zum Erstellen des Zertifikats

```
C:\Path_To_JDK\bin>keytool -genkey -keystore codeicare.keystore -alias  
codeicare -keyalg RSA -validity 10000
```

Die oben angegebenen Parameter werden in der nachstehenden Tabelle grob erklärt.

**Tabelle 9: Parameter zum signieren**

-genkey	Der Parameter um anzuzeigen dass ein Zertifikat erstellt wird.
-keystore [name]	Der Name des Zertifikats
-alias [name]	Der Alias name für das Zertifikat
-keyalg [alg]	Welcher Algorithmus wird für das Zertifikat verwendet?
-validity [val]	Wie lange soll das Zertifikat gültig sein?

Nach der Eingabe des obgenannten Befehls, wird man nach einem Passwort gefragt. Gefolgt von persönlichen Informationen, welche für das Zertifikat relevant sind. Folgende Einstellungen wurden für die CodeIcare Applikation verwendet.

**Tabelle 10: Zertifikatsdetails**

Passwort:	Codeicare09
Vor-, Nachname	Stefan Theler
Organisatorische Einheit:	Institute Icare
Organisation	Institute Icare
Ortschaft	Sierre
Kanton	Valais
Land	CH

Das (technische) Resultat sieht folgendermassen aus:

Zertifikatsdetails

```
CN=StefanTheler, OU=InstituteIcare, O=Institute Icare, L=Sierre,  
ST=Valais, C=CH.
```

Nachdem das Zertifikat erstellt wurde, kann die eigentliche Software signiert werden.

Befehl zum Signieren der

```
C:\Path To JDK\bin>jarsigner -verbose -keystore codeicare.keystore  
codeicare.apk codeicare
```

Der obenstehende Befehl führt diese Aufgabe aus und überschreibt unsignierte Applikation. Wie die Applikation später veröffentlicht im Android Market veröffentlicht wird, ist im Kapitel 8.2 genauer beschrieben.

## 7 Die Zukunftsaussichten

### 7.1 Allgemeine Technische Verbesserungen

Man kann eine Applikation immer verbessern, so auch CodeIcare. In 8 Wochen Entwicklungszeit kann keine Software entwickelt werden die ohne Fehler läuft und zudem noch stabil ist. Der Code kann immer verbessert und angepasst werden.

### 7.2 Update auf Android „Donut“

Gerüchten zu Folge soll ein Update der Android Software kurz bevor stehen. Darin soll evtl. auch eine überarbeitete Kamera-Version erscheinen<sup>1</sup>. So kann die Software vielleicht in dieser Hinsicht noch angepasst werden.

### 7.3 Der Verlauf / Die History

#### Anzeigen der Namen

Der Verlauf wurde als optionaler Punkt in das Projekt eingebunden und demnach noch nicht zu 100% fertig entwickelt. Der Verlauf sollte noch so weit angepasst werden, dass nicht die einzelnen Barcodes als Titel erscheinen, sondern die Namen der Produkte.

#### Löschen eines einzelnen Eintrages

Ein weiterer Punkt, der evtl. eingebaut werden kann, ist da Löschen eines einzelnen Eintrages. Zurzeit kann nur der gesamte Verlauf geleert werden. Benutzer können

#### Nur Tage anzeigen, welche Einträge haben

Zurzeit werden alle Tage angezeigt, auch wenn sie keine Einträge besitzen. Dies könnte manche Benutzer verwirren und sieht nicht wirklich schön aus. Der Übersicht halber sollte dies in einer nächsten Version angepasst werden.

### 7.4 Ausbau des Sprachinterfaces

Möglicherweise können mehrere Sprachen eingefügt werden. Für dieses Vorgehen ist ein sehr geringer Programmieraufwand nötig. Einzig das Resources-File muss erstellt und in den Präferenzen eingebunden werden. Zudem muss bei der Wahl der Sprache (in der Kamera-Aktivität) die neue Sprache angegeben werden.

---

<sup>1</sup> <http://www.journaldugeek.com/2009/07/27/quelques-captures-pour-android-donut/>

## 7.5 Local-Search

Damit die Benutzer wissen wo sie die Produkte kaufen können, wäre eine GPS-Lösung mit einer integrierten Google Maps-Suche hilfreich. Technische wäre dies eigentlich sehr einfach zu realisieren, da mit relativ wenig Aufwand auf die GPS Daten des Gerätes zugegriffen werden kann.

## 7.6 Werbung

In einer späteren Version der Applikation soll es möglich sein Werbung zu platzieren. Durch AdMob<sup>1</sup> ist es sehr einfach, Werbefbanner zu generieren und diese als Layer auf seine eigene Applikation zu legen. Der Entwicklungsaufwand würde sich Dank bereits existierenden Tutorials<sup>2</sup> auf ein Minum beschränken.

---

<sup>1</sup> <http://www.admob.com/>

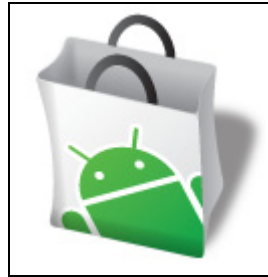
<sup>2</sup> <http://learningandroid.org/tutorial/2009/02/adding-admob-your-activity>

## 8 Android Market

### 8.1 Einführung

#### Was ist der „Android Market“

Abbildung 17: Android Market



Quelle:

<http://android-developers.blogspot.com>

Der Android Market ist eine Applikation, welche auf einem Android fähigen Smartphone installiert werden kann. Mit Hilfe des Android Markets ist es möglich, verschiedenen Applikationen, Spiele oder auch Hintergründe herunterzuladen und lokal auf dem Smartphone zu installieren.

Zudem ermöglicht der Android Market, das Veröffentlichen von eigenen Applikationen. Zur Zeit der Lancierung des Android Markets konnte man nur kostenlose Applikationen veröffentlichen. Nach einigen Monaten (Anfang 2009) eröffnete sich den Entwicklern aus den

Vereinigten Staaten, sowie aus Grossbritannien die Möglichkeit, ihre Applikationen zu verkaufen<sup>1</sup>.

Einige Wochen nach dem Release in den USA und Grossbritannien wurde der Upload von kostenpflichtige Applikationen zudem auch für Personen, welche in den folgenden Ländern ihren Wohnsitz (ihr Konto) besitzen, ermöglicht:

- Deutschland
- Frankreich
- Niederlande
- Österreich
- Spanien

Informationen, wann kostenpflichtige Applikationen auch aus der Schweiz angeboten werden können, sind zu diesem Zeitpunkt (Stand 15. Juli 2009) nicht bekannt. Eine offizielle Liste, in welchen Ländern der Verkauf von Software möglich ist findet man hier<sup>2</sup>.

---

<sup>1</sup> <http://android-developers.blogspot.com/2009/02/android-market-update-support-for.html>

<sup>2</sup> <http://www.google.com/support/androidmarket/bin/answer.py?hl=en&answer=138294>

## Android Market Timeline

**Tabelle 11: Android Market Timeline**

28. August 2009	Der Android Market wurde erstmals öffentlich erwähnt. <sup>1</sup>
22. Oktober 2008	Der Android Market ist ab dem 22. Oktober 2009 offiziell eröffnet und zugänglich für alle Benutzer <sup>2</sup>
13. Februar 2009	Ab diesem Zeitpunkt ist es möglich, kostenpflichtige Applikationen zu veröffentlichen. <sup>3</sup>

## 8.2 Die Veröffentlichung von Applikationen

### Registrierung

Damit man Android Applikationen veröffentlichen kann, ist eine einmalige Registrierung notwendig. Nach diesem Schritt wird der Benutzer aufgefordert einen eine Anmeldegebühr von \$25 zu bezahlen. Die Anmeldegebühr fungiert in diesem Sinne eher als Hemmschwelle, als dass sie als Einnahmequelle für Google dienen soll. Das Unternehmen begründet die Anmeldegebühr wie folgt:

*« There is a one time \$25 registration fee. We charge this fee to encourage higher quality products on the market (e.g. less spammy products) »* (Google, 2008)

Mit der Anmeldegebühr kann die Anzahl der Applikationen auf dem Android Market eingeschränkt werden. Erfahrungsgemäss sind viele Applikationen mangelhaft programmiert oder werden nicht genügend gewartet. Auch die Veröffentlichung sogenannter Spamming-Produkte kann so eingeschränkt werden. Es ist allgemein bekannt, dass Spamer ihre Tätigkeiten reduzieren, falls der Service nicht kostenlos ist.

Das Bezahlen der Anmeldegebühr muss direkt über Google Checkout erfolgen. Es sind keine weiteren Zahlungsmethoden möglich.

---

<sup>1</sup> <http://android-developers.blogspot.com/2008/08/android-market-user-driven-content.html>

<sup>2</sup> <http://android-developers.blogspot.com/2008/10/android-market-now-available-for-users.html>

<sup>3</sup> <http://android-developers.blogspot.com/2009/02/android-market-update-support-for.html>



## Veröffentlichung<sup>1</sup>

Um eine Applikation zu veröffentlichen muss man unbedingt folgende Voraussetzungen erfüllen:

- Die Applikation muss mit einem privaten Schlüssel signiert sein.
- Im `manifest.xml` muss der Versions-Code, sowie der Name der aktuellen Version eingegeben werden (`android:versionCode` und `android:versionName`)
- Im Manifest muss zudem ein Icon und eine „Inscription“ angegeben werden. (`android:icon` und `android:label`)

Da Entwickler oder Unternehmen aus der Schweiz zurzeit (Stand 15. Juli 2009) keine kostenpflichtige Applikationen veröffentlichen können, kann nach erfolgreichem Login auf <http://market.android.com/publish/> direkt den Button „*Upload Application*“ klicken. Das nun erscheinende Formular ist vollständig auszufüllen und zu bestätigen:

**Tabelle 12: Market Registration - Upload Assets**

Application .apk file	Die kompilierte Android Applikation. Nach der Auswahl des Files wird dieses in Echtzeit geprüft und angezeigt, ob und welche Rechte die Applikation benötigt. Diese Berechtigungen können z.B. Internet-Access oder GPS-Access betreffen und müssen vom Nutzer bestätigt werden, ansonsten kann er die Applikation nicht verwenden.
-----------------------	---

**Tabelle 13: Market Registration - Listing Details**

Language	Alle in der Applikation unterstützten Sprachen müssen hier angegeben werden. Für jede Sprache ist ein Titel und eine Beschreibung anzugeben.
Application Type	Um was für eine Applikation handelt es sich. Android unterscheiden zwischen zwei Objekte. Spielen und Applicationen.
Category	Nach der Auswahl des Types, ist die Kategorie zu wählen. Sie muss möglichst genau mit der Applikation übereinstimmen.
Price	Dieses Feld ist deaktiviert, da in der Schweiz keine kostenpflichtigen Applikationen vertrieben werden können.

<sup>1</sup> <http://developer.android.com/guide/publishing/publishing.html>

Tabelle 14: Market Registration - Publishing Options

Copy Protection	Zwei Optionen sind wählbar: Off: Die Applikation darf nach dem Download frei kopiert werden On: Verhindert das Kopieren nach dem Download. Verwendet mehr Speicherplatz auf dem Handy. Ist der Kopierschutz einmal aktiviert, kann er nicht mehr deaktiviert werden.
Locations	In welchen Länder soll die Applikation verfügbar sein. Falls man keine Einschränkungen angeben möchte ist die Wahl von „ <i>All Current and Future Locations</i> “ zu tätigen.

Tabelle 15: Market Registration - Contact Information

Website	Entwicklerwebseite
Email	E-Mail Adresse des Entwicklers.
Phone	Telefonnummer des Entwicklers

Nachdem alle Angaben eingegeben wurden, muss der Entwickler die Lizenzvereinbarungen, die sogenannten „*Android Content Guidelines*“ akzeptieren.

Da die Server von Google in der USA stehen und ein Herunterladen einer Software von einem amerikanischen Server einen Export darstellen, gelten hier die Exportbestimmungen der USA. Mehr Informationen zu diesen Bestimmungen können unter folgender Adresse bezogen werden:  
<http://www.google.com/support/androidmarket/bin/answer.py?answer=113770>.

Durch das Anwählen des dazugehörigen Buttons erklärt man sich mit diesen Exportbestimmungen einverstanden.

Die Software kann nun entweder direkt veröffentlicht oder erst zwischengespeichert werden. Der Unterschied ist, dass beim Speichern die Software noch nicht freigegeben wird. Das heißt, man kann später noch Änderungen vornehmen und erst danach veröffentlichen.

Durch Klick auf „*Publish*“ wird die Software direkt veröffentlicht und die Android-Gemeinschaft kann diese nun herunterladen und installieren.

Eine Übersichtsseite zeigt an, wie häufig die Software bereits installiert wurde. Zudem wird den Entwicklern angezeigt, wie die Android-Gemeinde die Applikation gewertet hat.

## Update

Nach dem Upload und der Veröffentlichung der Applikation kann es sein, dass die Entwickler zu einem späteren Zeitpunkt ein Update anbieten wollen. Hierzu muss man einfach die vorhandene Applikation ersetzt und eine neuere Version (welche im `AndroidManifest.xml` File angegeben werden muss) hochgeladen werden. Die Nutzer, welche die Software installiert haben, erhalten in der Benachrichtigungsleiste automatisch eine Information.

Nach Bestätigung wird die Software automatisch heruntergeladen und installiert.

## 8.3 Konkurrenz

### Apple AppStore

Der AppStore von Apple ist ein Store für das iPhone<sup>1</sup> und den iPod Touch<sup>2</sup>. Ähnlich wie beim Android Market ist der AppStore auch eine eigenständige Applikation und kann auf einem iPhone oder einem iPod Touch verwendet werden. Über die Applikation kann verschiedene (kostenlose und kostenpflichtige) Software bezogen werden. Um im AppStore zu navigieren, ist die Installation von iTunes erforderlich.

Die Produkte im AppStore sind kategorisch geordnet und können verschiedentlich sortiert werden. In den jeweiligen Kategorien sieht man auf einen Blick, welche Applikationen am meisten bezogen wurden.

Um Objekte zu laden ist ein iTunes-Store Konto erforderlich. Im Gegensatz zum Android Market ist es nicht möglich mit Google Checkout zu bezahlen. Für die Nutzung des AppStores muss eine Kreditkarte angegeben werden.

Die Applikationen können direkt über iTunes gekauft und geladen werden und mittels Synchronisierung auf das Gerät transferiert werden. Ebenfalls ist ein direktes Bezahlen und Herunterladen von Produkten über das Gerät möglich.

### Nokia Ovi Store

Der Nokia Ovi Store ist ein Teil des Produktes „Nokia Ovi“. Ähnlich wie die der Android Market und der App Store von Apple ist Nokia Ovi Store eine Web- und Handy-Applikation zugleich. Die Nokia Ovi Applikation ist mit über 100 Nokia Handymodellen kompatibel und deckt somit die Mehrheit der umfangreichen Nokia Palette ab.

---

<sup>1</sup> <http://www.apple.com/iphone/>

<sup>2</sup> <http://www.apple.com/ipodtouch/>

Neben dem Store stellt „Ovi“ (auf Deutsch „Tür“), eine Suite, ein Synchronisations-Tool, eine Mail-Applikation verschiedene andere Applikationen zur Verfügung.

Der Store wurde erst im Mai 2009 der breiten Öffentlichkeit vorgestellt aber besitzt bereits nach einem Monat ein umfangreiches Angebot. Zum Launch der Applikation wurden 20'000 Applikationen, Spiele und Widgets angeboten.

Da Nokia Handymodelle mit verschiedenen Betriebssystemen besitzen, ist es wichtig, dass der Store auch all diese abdeckt. Zurzeit sind Applikationen für das Symbian OS S60 und Nokia Series verfügbar.

Besitzer eines Nokia Handys oder Smartphones sind jedoch, im Gegensatz zu iPhone Kunden, nicht auf den eigenen Store angewiesen. Sie können Applikationen und Spiele weiter von Drittanbieter herunterladen und installieren. Ähnlich wie bei den anderen Stores, werden auch hier kostenlose und kostenpflichtige Applikationen. Mittels Kreditkarte kann man die Objekte direkt online bezahlen.

## 8.4 Vergleich Android Market vs. iPhone App Store

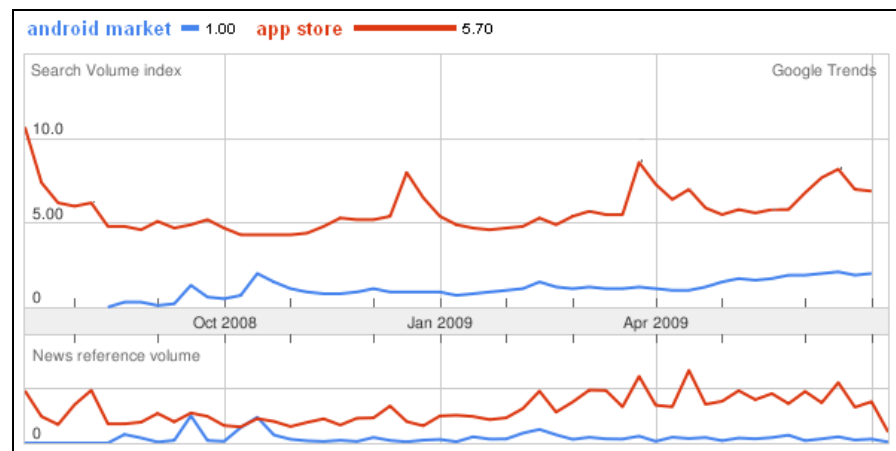
### Google Trends

Google Trends ist ein Analyse Tool um verschiedene Google-Suchbegriffe nach ihrer Häufigkeit zu ordnen. Durch die Angabe von verschiedenen Suchbegriffen kann man diese über einen gewissen Zeitraum miteinander vergleichen.

Google Trends kann natürlich nicht für wissenschaftliche Analysen verwendet werden, da sie nur Google-Suchbegriffe berücksichtigen. Doch weil Google-Suchanfragen erfahrungsgemäss kongruent mit dem Interesse der Internetgemeinde ist, kann Google Trends für die Analyse in dieser Arbeit sehr gut verwendet werden.

Die auf Google Trends basierte Grafik zeigt den Verlauf der Suchanfrage zwischen Juli 2008 und Juli 2009. Untersucht wurden die Suchbegriffe « Android Market » und « App Store ».

Abbildung 18: Such Volumen Index App Store und Android Market



Quelle: <http://trends.google.com>

Aus der Grafik geht hervor, dass die Nachfrage nach dem App Store immens grösser ist als die nach dem Android Market. Man sieht auch, dass der Android Market erst ab der zweiten Hälfte 2008 ein Thema wurde. Dies passt genau in die Timeline des Android Market. Am 28. August wurde dieser der breiten Öffentlichkeit vorgestellt.

Auch die News-Schlagzeilen, welche man im unteren Teil der Grafik betrachten kann, spricht eine eindeutige Sprache. Der iPhone App Store war viel häufiger in den Medien präsent als der Android Market. Nur im Oktober 2008 erschienen in etwa gleiche viele Schlagzeilen. Dies auf Grund der Eröffnung des Android Markets.

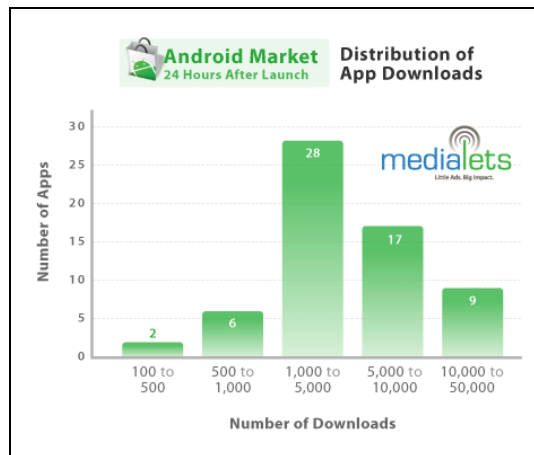
### Medialets-Vergleich nach den ersten 24h<sup>1</sup>

Medialets<sup>2</sup>, ein Unternehmen, welches sich auf Werbung in mobilen Applikationen spezialisiert hat, versuchte den Android Market und Apples AppStore nach den ersten 24h Online-Zeit zu vergleichen. Zwar ist dies schwierig, da der Android Market beim Release nur gerade 62 Applikationen / Spiele online gestellt hat. Dies ist 10% des Volumens, welches der AppStore in der gleichen Zeitspanne veröffentlicht hat.

<sup>1</sup> <http://www.medialets.com/blog/2008/10/23/android-market-vs-iphone-app-store-the-first-24-hours/>

<sup>2</sup> <http://www.medialets.com>

Abbildung 19: Downloadstatistik der ersten 24

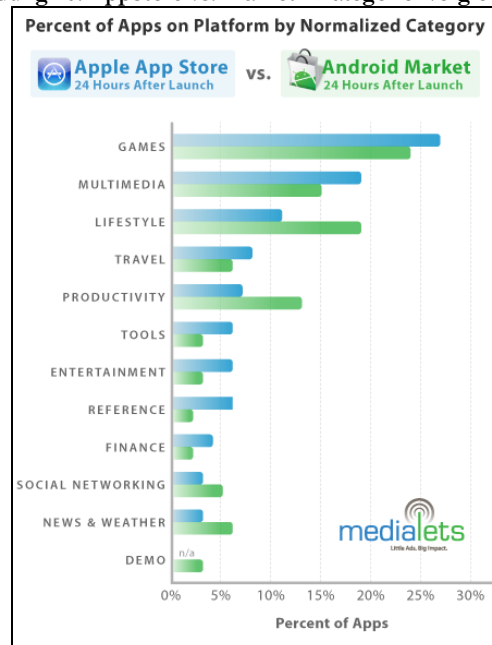


Quelle: <http://medialets.com>

Die Statistik zeigt auf, dass am meistens der 62 Anwendungen 1'000 bis 5'000 heruntergeladen wurden. 17 Applikationen wurden zwischen 5'000-10'000 mal heruntergeladen und nur deren 2 zwischen 100- und 500-mal. Diese Informationen werden, im Gegensatz zum AppStore jeweils direkt beim Downloadlink im Android Market angezeigt. Apple hat die Zahlen zur Häufigkeit ihrer Downloads nicht veröffentlicht.

Der direkte Vergleich ist also schwierig zu bewerkstelligen, da bei Apple zu Beginn ein riesiger Hype um das iPhone ausgebrochen ist. Android war am Anfang eher in Entwicklerkreisen bekannt. Die folgende Grafik versucht den prozentuellen Vergleich der beiden Stores miteinander zu vergleichen. Untersucht wurde die Anzahl der Applikation innerhalb einzelner Kategorien.

Abbildung 20: AppStore vs. Market - Kategorienvergleich



Quelle: <http://medialets.com>

Da die Kategorien des Android Markets und des AppStores nicht identisch sind, wurden diese von Medialets in einheitliche Kategorien umgewandelt und zusammengeführt.

Tabelle 16: Medialets Kategorienverwaltung

Diagramm	AppStore	Android Market
Games	Games	Games
Multimedia	Books, Music	Multimedia
Lifestyle	Lifestyle, Healthcare & Fitness, Sports, Photography	Lifestyle, Shopping
Travel	Travel, Navigation	Travel
Productivity	Productivity, Business	Productivity
Tools	Utilities	Tools, SW Library
Entertainment	Entertainment	Entertainment
Reference	Reference, Education	Reference
Finance	Finance	Finance
Social Networking	Social Networking	Social, Communication
News & Weather	News, Weather	News & Weather
Demo	N/A	Demo













So konnte festgestellt werden, welcher Anbieter wie viele Produkte in welcher Kategorie veröffentlichte.

Aus der Grafik sticht heraus, dass die Hauptaugenmerke der Entwickler auf den gleichen Kategorien beruhen. Einzig im Lifestyle- und Productivity-Bereich haben die Android-Entwickler in den ersten 24h prozentual deutlich mehr Applikationen veröffentlicht als die iPhone-Entwickler. Natürlich sind hier nur die 62 Android Applikationen berücksichtigt worden und somit nur ein Zehntel des iPhone Sortiments. Es ist jedoch gleichwohl interessant, dass die Entwickler bei beiden Plattformen von Beginn weg ähnliche Applikationen entwickelt wurden. Dies könnte natürlich auch an der Entwicklergemeinschaft liegen, welche möglicherweise dieselbe ist.



## 9 Schlussbericht

### 9.1 Gesteckte Ziele vs. Erreichte Ziele

Zieldefinition	
Die Applikation muss auf allen Smartphones, welche Android einsetzen funktionieren.	
Die Menüführung muss der bestehenden Software des RFID-Centers angepasst werden.	
Die Applikation erstellt ein Foto des Barcodes und decodiert dieses mit Hilfe der bereits erstellten Klasse des RFID-Centers.	
Die Applikation muss einen Browser öffnen können	
Die Applikation muss eine sichere Verbindung (HTTPS) zu einem Webserver herstellen.	
Die Daten müssen in einer internen Datenbank abgespeichert werden.	
Die Applikation muss alle gängigen Barcodes (1D, 2D) lesen und verarbeiten können.	
Der Barcode sollte auch mittels Videoaufnahme erkannt werden.	
Durch das verwenden des Autofokus soll ein unscharfes Foto des Barcodes verhindert werden.	
<i>Die Applikation wird über den Android Market vertrieben (Android Market Analyse)</i>	
<i>Speichern der Historie. Der Benutzer kann abrufen, welche Barcodes er bereits gescannt hat.</i>	
<i>Automatische Erkennung ob 1D Code oder 2D Code.</i>	

**Anmerkung:** Die Kursiv geschriebenen Einträge wurden zur Zeit der Zieldefinition als „Wunschkriterien“ aufgelistet.

Die gesteckten Ziele, die zu Beginn der Bachelorarbeit definiert wurden, sind praktisch vollständig erreicht worden. Die Verbindung zum Internet wird nicht durch eine gesicherte HTTPS-Verbindung hergestellt, da der Server des Instituts dies zurzeit noch nicht unterstützt. Zwar wurden bereits Klassen implementiert die dies unterstützen sollten, doch dies konnte aus obgenannten Gründen nicht getestet werden. Des Weiteren erkennt der Barcodeleser nur EAN-Codes und Datamatrizen. QR-Codes und weitere Barcodetypen werden zurzeit noch nicht

unterstützt, da das RFID-Center das Erkennen dieser Typen noch nicht in ihre Decoder-Klasse integriert wurde. Zu einem späteren Zeitpunkt wird dies noch nachgeführt.

Alle anderen, zum Teil zusammenhängende Ziele, wurden in der vorgegebenen Frist erreicht.

## 9.2 Known Bugs

### Decodierungszeit

Im Vergleich zur iPhone Applikation beansprucht die Decodierungszeit in der Android-Version immer noch um einiges mehr Zeit. Eine Fehlerquelle konnte bereits ausfindig gemacht werden und wird in der Version 2.0 behoben. Dies ist die unnötige Transformierung von YUV420 in Bitmap-Daten und danach wieder in ARGB Daten. Die Entwickler des Icare Institutes programmieren bereits jetzt an einer Lösung für dieses Problem.

Um die Decodierungszeit so niedrig wie möglich zu halten ist die genaue Analyse der verschiedenen Methoden notwendig.

### Funktionsabstürze

Aus unerklärlichen Gründen kann es vorkommen, dass die Kamera Activity abstürzt und nicht mehr weiterarbeitet. Falls dieses Problem auftritt muss das Handy neu gestartet werden, weil auch die Standard-Kamera Applikation auf die gleichen Ressourcen zugreifen und somit nicht mehr gestartet werden kann. Gegen Ende der Arbeit konnte das Problem zwar etwas eingeschränkt werden und es trat nicht mehr auf, doch ganz ausschliessen kann man dieses Problem noch nicht.

## 9.3 Persönlicher Kommentar des Autors

Für mich war ein Projekt in dieser Grössenordnung neu und demnach sehr spannend zu realisieren. Zwar wurden wir mittels kleineren, in einem Team durchgeführten Projektarbeiten sehr gut auf die Bachelorarbeit vorbereitet, doch mit der Realisierung dieser Bachelorarbeit nicht zu vergleichen. Ein von Grund auf selber gestaltetes Projekt erfordert sehr viel Selbstdisziplin und Engagement. Zudem muss natürlich das nötige Know-how stimmen, damit man die gesteckten Ziele erreichen kann.

Mir persönlich hat die Realisierung, der hier vorliegenden Arbeit, jederzeit Spass und Freude bereitet. Durch die hervorragende Zusammenarbeit mit dem Icare Institut und dem sehr sorgfältig geplanten Projektablauf konnten alle Schritte fristgerecht durchgeführt und realisiert werden.

Herr Laurent Sciboz, Verantwortlicher Dozent, hielt sich eher im Hintergrund auf und liess mich sehr frei arbeiten, war jedoch für allfällige Fragen jederzeit verfügbar. Durch die gemeinsame Erstellung des Pflichtenhefts wurde rasch klar, welche Ziele erreicht werden sollen und dementsprechend früh war ein produktives Arbeiten möglich.

Mit dem Ablauf und dem Resultat meiner Arbeit bin ich im Grossen und Ganzen sehr zufrieden. Es ist klar, dass noch einige Verbesserungen durchgeführt werden könnten, doch dies ist wohl bei jeder Applikation der Fall.

Die verschiedenen Projekt-Schritte verliefen jeweils praktisch ohne Komplikationen, oder grössere Probleme. Einzig der Einbau des C-Decoders stellte sich als komplexer dar als gedacht. Doch durch die grosszügige Unterstützung von Herrn Christophe Abbet konnte auch dieses Problem relativ rasch gelöst werden.

In der Statistik der Projektplanung (zu finden im Anhang) sieht man, dass die Planung im Grossen und Ganzen sehr gut gestimmt hat. Es gab keine Bereiche, die viel mehr Zeit in Anspruch genommen haben, als geplant. Durch die knapp bemessene Zeit, ist am Schluss die Qualitätsprüfung jedoch ein wenig vernachlässigt worden. Zum Teil wurden diese Qualitätstests auch parallel zur Entwicklung vom Institut Icare durchgeführt und mir anschliessend die Resultate und Änderungswünsche durchgegeben. Die Tests auf verschiedenen Android Handy-Modellen konnten ebenfalls nicht durchgeführt werden, da keine anderen Modelle zur Verfügung standen.

## Literaturverzeichnis

Developerlife. (Juli 2008). *Developerlife*. Abgerufen am 1. August 2009 von Developerlife: <http://developerlife.com/tutorials/wp-content/uploads/2008/07/bootcamp.pdf>

Developers, A. (kein Datum). *Android Developer Webseite*. Abgerufen am 1. August 2009 von Android Developer Webseite: <http://developer.android.com>

Gibara, T. (kein Datum). *Tom Gibara*. Abgerufen am 1. August 2009 von Tom Gibara: <http://www.tomgibara.com/android/camera-source>

Google. (2008). <http://market.android.com>. Abgerufen am 2. Juni 2009 von Android Market - Website: <http://market.android.com/support/bin/answer.py?hl=en&answer=113468>

Kistler, R. (kein Datum). *Ruoss-Kistler*. Abgerufen am 1. August 2009 von Ruoss-Kistler: <http://www.ruoss-kistler.ch/Frameload.htm?http://www.ruoss-kistler.ch/barcode.htm>

Lee, W.-M. (kein Datum). *DevX.com*. Abgerufen am 1. August 2009 von DevX.com: <http://www.devx.com/wireless/Article/39972/1763/page/1>

Medialets. (31. Oktober 2008). *Medialets*. Abgerufen am 1. August 2009 von Medialets: <http://www.medialets.com/blog/2008/10/31/android-market-unleashed/>

Meier, R. (2009). *Professional Android Application Development*. Indianapolis, IN 46256: Wiley Publishing, Inc

netmite.com. (2008). *JNI Tips*. Abgerufen am 1. August 2009 von JNI Tips: <http://www.netmite.com/android/mydroid/dalvik/docs/jni-tips.html>

Padawan, J. (24. November 2008). *Android PH*. Abgerufen am 1. August 2009 von J2ME - Android : <http://www.androidph.com/2008/11/camera-capture.html>

plusminus. (10. Januar 2008). *AndDev*. Abgerufen am 1. August 2009 von AndDev: [http://www.anddev.org/doing\\_http\\_post\\_with\\_android-t492.html](http://www.anddev.org/doing_http_post_with_android-t492.html)

Team, Z. (2008). *ZXing*. Abgerufen am 1. August 2009 von ZXing: <http://code.google.com/p/zxing/>

## Anhang 1 – Die Wochenrapporte

### Woche 1

#### Erstellung der Planung und des Pflichtenheftes

*Das Raster der Planung wurde von einem vorherigen Projekt übernommen und ein wenig angepasst. Um einen Überblick zu erhalten wurde zuerst das Pflichtenheft erstellt. So konnte man bereits sehen welche Aufgaben erledigt werden müssen und welche Priorität diese haben. Nach dem ersten Entwurf des Pflichtenheftes habe ich die Planung erstellt.*

**4h**

#### Installation und Konfiguraton der Entwicklungsumgebung

*Da es ein ziemlich detailliertes Tutorial auf der offiziellen Android Webseite gibt, konnte dieser Punkt ohne Probleme und ziemlich schnell durchgeführt werden.*

**2h**

#### Sammeln von Informationen und Beginn mit Beispielprojekten

*Am Anfang der Bachelorarbeit ist es wichtig, dass man sich eine grosse Ansammlung von Büchern, Artikel, Beispielen etc. anschafft. Dieser Punkt war zu Beginn der ersten Woche sehr wichtig.*

**16h**

#### Einarbeit in die Android SDK und in das Thema

*Die SDK ist ziemlich komplett, darum muss man sich genau in den Development Guide einlesen und Informationen über die Machbarkeit der verschiedenen Teile der Bachelorarbeit einholen.*

**4h**

#### Total Stunden:

**22h**

#### Diverses

Ab sofort jeden Montag um 13.30 Uhr Meeting mit Jean-Jérôme Sarrasin im Showroom des RFID-Centers.

## Woche 2

### Projektverwaltung und Sitzung

*Nachdem die Planung und das Pflichtenheft mit Jean-Jérôme besprochen wurde, habe ich noch einige Änderungen gemacht und Ergänzungen hinzugefügt. Auch hinsichtlich der Planung hatten diese Änderungen Konsequenzen, so mit musste auch diese angepasst werden.*

**2h**

### Pflichtenheft und Planung abgeändert und erweitert

*Nachdem die Planung und das Pflichtenheft mit Jean-Jérôme besprochen wurde, habe ich noch einige Änderungen gemacht und Ergänzungen hinzugefügt. Auch hinsichtlich der Planung hatten diese Änderungen Konsequenzen, so mit musste auch diese angepasst werden.*

**5h**

### Beispielprojekte

*Um die nötigen Android Kenntnisse zu erhalten ist es wichtig, dass man am Anfang viele Tutorials und Beispiele durchführt. Dieser Punkt war in dieser Woche*

**8h**

### Sammeln von Informationen

*Einen Teil der Woche wurde wieder dem Sammeln von Android Informationen gewidmet. Durch die Besprechung mit Jean-Jérôme sind einige Punkte klarer geworden. So konnte man sich bei der Suche nach Informationen auf spezifisch benötigte Funktionen beschränken.*

**3h**

### Dokumentation

*Die Hauptdokumentation wurde durch die Punkte „Android – Erste Schritte“ und „Android – Architektur“ erweitert.*

**2h**

### Total Stunden:

**20h**

### Diverses

Die Sitzungen mit Jean-Jérôme werden ab sofort um 1430 Uhr anstatt 1330 Uhr (montags) stattfinden.

Das Pflichtenheft muss noch signiert werden. Es ist nicht nötig, dass alle Wochenreporte per Mail an Herrn Sciboz geschickt werden. Es reicht, wenn diese gewissenhaft erstellt und abgelegt werden.

## Woche 3

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Wöchentliche Sitzung mit Jean-Jérôme am Montag durchgeführt. Wochenplanung und Deklaration der Wochenstunden geschrieben.*

**2h**

### Dokumentation

*Die allgemeinen Informationen zu Android (Was ist Android nicht?, Architektur) wurden weitergeführt.*

**4h**

### Konkrete Einarbeit in das Thema

*Beispiele für die Menüführung und Formulare programmiert.  
Es wurde mit Beispielprojekten für das Kamerahandling begonnen.*

**14h****Total Stunden:****20h**

Schwierigkeit	Warum aufgetreten?	Lösung
Kamerahandling	Beispielcode im Internet ist oftmals noch veraltet (Alte SDK).	-

### Diverses

Nächste Sitzung mit Jean-Jérôme anstatt Montag, 01.06.2009 am 03.06.2009 um 1030 Uhr. Die Sitzung am 15. Juni wurde gestrichen (Grund : Jean-Jérôme Ferien und ich Modulprüfungen)

## Woche 4

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Wöchentliche Sitzung mit Jean-Jérôme am Mittwoch durchgeführt. Wochenplanung und Deklaration der Wochenstunden geschrieben.*

**2h**

### Android Market / Dokumentation

*Einarbeit und Recherche des Android Markets. Suche nach Informationen. Dokumentation durch den Punkt Android Market erweitert.*

**12h**

### Einarbeit in Decoder Java Klasse

*Nach dem Erhalten der Klasse wurde diese ein erstes Mal untersucht und analysiert.*

**1h****Total Stunden:****20h**

Schwierigkeit	Warum aufgetreten?	Lösung
Android Market Analyse	Da ich das Smartphone noch nicht erhalten habe, konnte ich nicht alle Teile analysieren.	Nach Erhalt des Smartphones den Punkt Android Market in der Dokumentation erweitern. Erwarteter Aufwand: 4h

### Diverses

Durch den Feiertag (Pfingstmontag) wurde diese Woche nicht 20 Stunden gearbeitet. Die fehlenden Stunden werden während der Ferienwoche kompensiert.



## Woche 5

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Da die letzte Sitzung am Ende der letzten Woche war, wurde das Meeting der heutigen Woche auf den Mittwoch verschoben. Am Mittwoch traf das Smartphone ein und wurde mir übergeben.*

**2h**

### Analyse der Java Decoder Klasse

*Damit ich verstehe, welche Operationen in der erhaltenen Klasse von statten gehen, habe ich diese analysiert und getestet.*

**2h**

### Erstellung der Menüs und Funktionen

*Um die verpassten Stunde der letzten Woche zu kompensieren, wurde bereits mit der Erstellung des Menüs und den verschiedenen Funktionalitäten begonnen. Die Vorlagen für die „About“ Seite und die Konfiguration wurden erstellt.*

**10h**

### Locales

*Ein wichtiger Punkt der Applikation ist, dass sie in mehreren Sprachen verfügbar sein muss. Diese Sprache muss individuell von der globalen Smartphone-Sprache ausgewählt werden können. Durch das Erstellen von individuellen Resource-Folders konnte bereits die Standardsprache abgedeckt werden. Die Lösung für manuelle Auswahl wurde noch nicht implementiert.*

**4h**

### Kamera Handling

*Um ein Gefühl für das Kamera Handling zu bekommen wurde eine kleine Test Applikation, welche auf ein Sample – Projekt der Android Gemeinschaft basiert, erstellt. Die Test Applikation wurde so programmiert, dass Sie später in das Projekt übernommen werden kann.*

**10h****Total Stunden:****28h**

Schwierigkeit	Warum aufgetreten?	Lösung
Keine		

### Diverses

Die Fertigstellung der Android Market Analyse wurde auf das Ende der Projektarbeit verschoben. Es ist nun wichtiger, dass die Applikation entwickelt wird. Woche 6 finden die Abschlussprüfungen statt und wird deshalb nicht für die Bachelorarbeit verwendet.

## Woche 7

### Projektverwaltung (Sitzungen / Planung / Administratives)

*In der Sitzung am Montag wurde eine Änderung der Planung vorgenommen. Das Ziel war es, dass der Decoder am Freitag ein erstes Mal vorgestellt werden kann. Also wurde abgemacht, dass nicht die Konfiguration und das Menü diese Woche entwickelt werden, sondern dass die Konzentration 100% auf das Barcodehandling liegt*

3h

### Kamera und Barcodehandling

*Das Ziel dieser Woche war es, dass Barcodehandling grösstenteils abzuschliessen. Doch dies konnte nicht realisiert werden. Zwar funktionierte der Decoder bis Ende der Woche. Doch die Decodierungszeit von über einer Sekunde war zu lange. So musste man eine neue Lösung suchen.*

30h

### Erstellung der Menüs und Funktionen

*Die TestCamera-Applikation wurde gegen Ende der Woche in das Hauptprojekt eingefügt. Damit der Benutzer wählen kann, ob er Matrizen oder lineare Barcodes scannen möchte.*

2h

**Total Stunden:**
**35h**

Schwierigkeit	Warum aufgetreten?	Lösung
Absturz der Applikation	Die Applikation wird wohl nicht richtig beendet, Threads laufen weiter oder irgendwelche andere Fehler treten beim Kamerahandling auf.	Debugging und Fehlersuche
Decodierung zu langsam	Die Java Klasse des RFID-Centers, welche die Decodierung des Fotos sicherstellt, arbeitet zu langsam. C++ Decoderklasse wäre schneller	Versuchen die C++ Klasse in das Android Projekt einzubauen.

### Diverses

Am Donnerstag wurde bestimmt, dass ich für 2 Tage an einer JNI-Lösung arbeite. Das heisst, es soll möglich sein einen C++-Decoder in die Java Applikation einzubauen. Falls mir dies nicht gelingt, soll ich nicht zu viel Zeit darin investieren.

## Woche 8

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Sitzung am Montag, Planung und ToDo Liste angepasst.*

**2h**

### Kamera und Barcodehandling

*Nach weiterer Recherche habe ich festgestellt, dass ein Android NDK (Native Development Kit) freigegeben wurde. Mit dem NDK ist es möglich, dass C Klassen eingebunden werden. Die Analyse und das Testen der NDK stand diese Woche im Vordergrund. Bis zum Ende der Woche wurde noch keine Lösung des Problems gefunden.*

**25h**

### Erstellung der Menüs und Funktionen

*Das Menü wurde um den Punkt „Registration“ erweitert. Dieser Eintrag dient nur zu Debugzwecken und wird später wieder entfernt. Hier kann die Registrierung eines Users getestet werden.*

**8h**

### Sicherheit

*Momentan kann auf den Server nur mittels HTTP zugegriffen werden. Später soll jedoch ein Zugriff mittels HTTPS ermöglicht werden. Dies musste bereits integriert werden.*

**4h**

### Datenverarbeitung

*Nachdem ein Barcode eingescannt wird, soll die Applikation Daten von einem Webserver holen. Dies wurde in einem Tag komplett realisiert. Funktionsfähig sowohl für Manuelles Input, sowie Scanning.*

**8h**

**Total Stunden:**

**47h**

Schwierigkeit	Warum aufgetreten?	Lösung
Absturz der Applikation	Die Applikation wird wohl nicht richtig beendet, Threads laufen weiter oder irgendwelche andere Fehler treten beim Kamerahandling auf.	Debugging und Fehlersuche (Wurde noch nicht gelöst)
Decodierung zu langsam	Die Java Klasse des RFID-Centers, welche die Decodierung des Fotos sicherstellt, arbeitet zu langsam. C++ Decoderklasse wäre schneller	Versuchen die C++ Klasse in das Android Projekt einzubauen. (Wurde noch nicht gelöst)

## Woche 9

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Sitzung am Montag, Planung und ToDo Liste angepasst.*

**2h**

### Kamera und Barcodehandling

*Das Barcodehandling bereitet weiter Probleme. Zusammen mit dem RFID-Center wurde weiter versucht die C-Klasse einzubauen. Immer noch ohne Erfolg.*

**15h**

### Erstellung der Menüs und Funktionen

*Das Menü, resp. die restlichen Funktionen wurden grösstenteils fertig gestellt und müssen nur noch getestet werden. Es fehlt noch die History und Anpassungen im Konfigurations-Bereich.*

**15h**

### Dokumentation

*Die Dokumentation wurde weitergeführt*

**4h**

**Total Stunden:**

**36h**

Schwierigkeit	Warum aufgetreten?	Lösung
Absturz der Applikation	Die Applikation wird wohl nicht richtig beendet, Threads laufen weiter oder irgendwelche andere Fehler treten beim Kamerahandling auf.	Debugging und Fehlersuche (Wurde noch nicht gelöst)
Decodierung zu langsam	Die Java Klasse des RFID-Centers, welche die Decodierung des Fotos sicherstellt, arbeitet zu langsam. C++ Decoderklasse wäre schneller	Versuchen die C++ Klasse in das Android Projekt einzubauen. (Wurde noch nicht gelöst)

### Diverses

Ich musste an zwei Tagen Vorstellungsgespräche in Bern und Zürich besuchen, deshalb wurde in dieser Woche weniger gearbeitet als sonst.

## Woche 10

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Sitzung am Montag, Planung und ToDo Liste angepasst.*

**3h**

### Kamera und Barcodehandling

*Das Barcodehandling bereitet weiter Probleme. Zusammen mit dem RFID-Center wurde weiter versucht die C-Klasse einzubauen. Immer noch ohne Erfolg.*

**4h**

### Sicherheit implementieren

*Der HTTP-Authentication Modus wurde weitergebaut.*

**4h**

### Erstellen der Menüs und Funktionen

*Die History wurde weiter programmiert und fast abgeschlossen. Beim Konfigurationsbereich wurden die Preferences eingebaut und fertig programmiert.*

**10h**

### Dokumentation

*Die Dokumentation wurde weitergeführt.*

**24h****Total Stunden:****45h**

Schwierigkeit	Warum aufgetreten?	Lösung
Decodierung	Das Decodierungsproblem wurde immer noch nicht vollständig gelöst. Wir sind jedoch zuversichtlich, dass dies in der nächsten Woche funktioniert.	Zusammen mit dem RFID-Center weiter an der Lösung arbeiten

### Diverses

Ich musste an zwei Tagen Vorstellungsgespräche in Bern und Zürich besuchen, deshalb wurde in dieser Woche weniger gearbeitet als sonst.

## Woche 11

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Sitzung am Montag, Planung und ToDo Liste angepasst.*

**1h**

### Kamera und Barcodehandling

*Das Decodierungs-Problem konnte nun zusammen mit dem RFID-Center gelöst werden. Nun werden die Codes viel schneller gescannt also bei dem Java Decoder. Der Umweg über die Erstellung eines Bitmaps braucht noch ein wenig zu viel Zeit, kann aber wegen Zeitmangels nicht mehr während dieser Diplomarbeit gelöst werden. Wurde auf später verschoben*

**15h**

### Erstellen der Menüs und Funktionen

*Ein Fehler in der History und in der Sprachwahl wurde behoben*

**5h**

### Dokumentation

*Die Dokumentation wurde weitergeführt.*

**20h****Total Stunden:****45h**

Schwierigkeit	Warum aufgetreten?	Lösung

### Diverses

Keine Sitzung, da Jean Jérôme verhindert war. Zusätzlich hatte ich am Freitagnachmittag noch ein Vorstellungsgespräch in Bern.

## Woche 12

### Projektverwaltung (Sitzungen / Planung / Administratives)

*Sitzung am Montag, Planung und ToDo Liste angepasst. Brennen der CD, organisieren des Druckes*

**3h**

### Kamera und Barcodehandling

*Das Decodierungs-Problem konnte nun zusammen mit dem RFID-Center gelöst werden. Nun werden die Codes viel schneller gescannt also bei dem Java Decoder. Der Umweg über die Erstellung eines Bitmaps braucht noch ein wenig zu viel Zeit, kann aber wegen Zeitmangels nicht mehr während dieser Diplomarbeit gelöst werden. Wurde auf später verschoben*

**15h**

### Qualitätsprüfung

*Die Applikation wurde getestet und danach konnten noch einige Anpassungen vorgenommen werden*

**5h**

### Erstellen der Menüs und Funktionen

*Nach den Tests wurden noch einige Änderungen in den Bereichen Design und Handling vorgenommen. Das Hauptmenü angepasst und die Icons neu definiert.*

**4h**

### Dokumentation

*Die Dokumentation wurde weitergeführt. Korrigieren und Rechtschreibung prüfen.*

**38h**
**Total Stunden:**
**50h**

Schwierigkeit	Warum aufgetreten?	Lösung

**Diverses**

## Woche 13

### Druck und Abgabe

*Bachelorarbeit gedruckt, gebunden und abgegeben*

**3h****Total Stunden:****3h**

Schwierigkeit	Warum aufgetreten?	Lösung



## Anhang 2 - Das Pflichtenheft

### Allgemeines

#### Allgemeine Beschreibung

In diesem Pflichtenheft wird in konkreter Form beschrieben, welche Ziele in dieser Bachelorarbeit zu erreichen sind. Ebenfalls wird der Rahmen für diese Zielfindung gestellt.

Das Pflichtenheft wird vom Student erstellt und zusammen mit dem Vertreter der Schule und des RFID-Centers ausgearbeitet.

#### Gültigkeit

Nach der Erfassung und Bestätigung durch den Auftraggeber (Vertreter der Schule, Vertreter des RFID-Centers) erhält das Pflichtenheft den Status „gültig“ und muss streng verfolgt werden.

Nachträgliche Änderungen des Pflichtenheftes müssen mit allen Teilnehmer (Student, Dozent, RFID-Center) abgesprochen werden.

#### Teilnehmer dieser Bachelorarbeit

Funktion	Person
Student	Stefan Theler
Vertreter der Schule / Verantwortlicher	Laurent Sciboz
Vertreter des RFID-Centers	Jean-Jérôme Sarrasin

#### Allgemeine Bestimmungen

In einer ersten Sitzung wurde bestimmt, dass die Bachelorarbeit auf Deutsch durchgeführt wird. Alle Dokumentationen, also die technische und die allgemeine Dokumentation werden auf Deutsch geschrieben.

Man einigte sich darauf, dass die Codekommentare und der Code auf Englisch erstellt werden. So ist es auch für Französisch sprechende Personen möglich den Code zu verstehen.

## Zielbestimmung

### Aufgabenstellung

Diese Bachelorarbeit wird im Auftrag der HES-SO Valais und dem RFID-Center entwickelt. Ziel ist es, einen bestehenden Barcodeleser auf das neuartige Smartphone-Betriebssystem Android zu portieren. Somit ist diese Bachelorarbeit in die Sparte „Innovation – Neue Produkte“ einzuordnen und zu bewerten.

Zudem soll für die spätere Nutzung eine technische Dokumentation, sowie eine allgemeine Beschreibung von Android Applikation erstellt werden.

### Musskriterien

- Die Applikation muss auf allen Smartphones, welche Android einsetzen funktionieren.
- Die Menüführung muss der bestehenden Software des RFID-Centers angepasst werden.
- Der Barcode muss über die Kamera des Smartphones erfasst werden.
- Die Applikation erstellt ein Foto des Barcodes und decodiert dieses mit Hilfe der bereits erstellten Klasse des RFID-Centers.
- Die Applikation muss einen Browser öffnen können
- Die Applikation muss eine sichere Verbindung (HTTPS) zu einem Webserver herstellen.
- Die Daten müssen in einer internen Datenbank abgespeichert werden.
- Die Applikation muss alle gängigen Barcodes (1D, 2D) lesen und verarbeiten können.
- Der Barcode sollte auch mittels Videoaufnahme erkannt werden.
- Durch das Verwenden des Autofocus soll ein unscharfes Foto des Barcodes verhindert werden.

### Wunschkriterien

- Die Applikation wird über den Android Market vertrieben (Android Market Analyse)
- Speichern der Historie. Der Benutzer kann abrufen, welche Barcodes er bereits gescannt hat.
- Automatische Erkennung ob 1D Code oder 2D Code.

## Produkteinsatz

### Anwendungsbereiche

Personen verwenden diese Software um Informationen zu Produkten / Dienstleistungen zu erhalten, um Preisvergleiche durchzuführen oder einfach

Informationen zu Örtlichkeiten von gewissen Produkten / Dienstleistungen erhalten wollen.

Ein Beispiel: Eine Person wünscht sich Informationen zu einem gewissen Produkt. Auf der Verpackung des Produktes ist nur ein Barcode ersichtlich. Nun kann die Person mit ihrem Smartphone den Barcode fotografieren und eine Webseite mit den Informationen zu diesem Produkt wird angezeigt. Diese Informationen können zum Beispiel Herkunftsland, Allergien oder auch Verfallsdaten haben.

## **Zielgruppen**

Die Applikation richtet sich an alle Personen, welche Preisvergleiche erstellen möchten, mehr Informationen zu Produkten / Dienstleistungen oder einfach wissen wollen, wo dieses Produkt zu kaufen ist.

Es kann von Manager, von Hausfrauen, von Bauarbeiter oder Kinder eingesetzt werden. Es gibt also keine Einschränkungen und ist an alle Personengruppen gerichtet.

Die einzige Voraussetzung für die Benutzung der Applikation ist, dass sie ein Smartphone, welches unter Android läuft, besitzt.

## **Betriebsbedingungen**

Die Applikation soll 24h, also rund um die Uhr, von dem Android Market bezogen werden können.

Einmal installiert, kann die Applikation so oft verwendet werden, wie der Benutzer will. Die Applikation soll wartungsfrei verwendet werden dürfen.

## **Programmfunktionen**

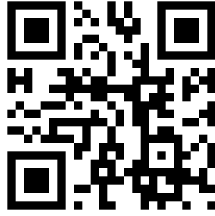
### **Funktionsübersicht**

Die Menüführung der Applikation soll genau gleich aussehen wie bei den bereits implementierten Barcodeleser des RFID-Centers.

- Read 2D Barcode
- Read 1D Barcode
- Manual Input
- On the Web
- About
- Configuratoin
- ( History )

## Einzelne Funktionen kurz erklärt

### /F0010/ Read 2D Barcode



Diese Funktion ist ein Hauptbestandteil der Applikation. Hier werden 2D Codes ausgelesen und mittels einer erhaltenen Java Klasse encodiert.

Wenn die Enkodierung erfolgreich war, greift die Applikation mittels Browser auf das Internet zu und öffnet eine Webseite mit Informationen zum Produkt oder zur Dienstleistung.

### /F0020/ Read 1D Barcode



Ähnlich wie bei /F0010/ liest diese Funktion einen Barcode ein und encodiert diesen mittels der erhaltenen Java Klasse. Der einzige Unterschied ist, dass diese Funktion einen 1D Barcode liest.

Wenn die Enkodierung erfolgreich war, greift die Applikation mittels Browser auf das Internet zu und öffnet eine Webseite mit Informationen zum Produkt oder zur Dienstleistung.

**Anmerkung:** Es ist noch nicht genau klar ob die Funktionen /F0010/ und /F0020/ zu einer einzigen Funktion zusammengebaut werden könne. So würde die Applikation selber entscheiden, um welchen Barcode Typ es sich handelt. In diesem Fall würden die beiden Funktionen also zu einer Funktion zusammengeführt.

### /F0030/ Manual Input

Falls ein Barcode einmal nicht gelesen werden kann, so kann diese Funktion aufgerufen werden. Hier kann man den Code manuell eingeben und so die Informationen abrufen.

### /F0040/ On the Web

Bei dieser Funktion kann der Benutzer Informationen über das RFID-Center erhalten. Wenn dieser Punkt angeklickt wurde, so öffnet sich ein Browserfenster mit der Webseite des RFID-Centers.

### /F0050/ About

Diese Funktion stellt lediglich einige Informationen zur Applikation dar. Hier kann man die aktuell installierte Version und das Impressum nachlesen

**/F0060/ Configuration**

Unter der Konfiguration kann man die Sprache der Applikation wählen. In einer ersten Phase werden Englisch und Französisch implementiert. Später sollen auch noch weitere Sprachen hinzugefügt werden können.

**( /F0070/ History )**

Diese Funktion zeigt auf, welche Anfragen ein Benutzer bereits abgeschickt hat. Hier kann man auf den Verlauf der vorangegangenen Barcodes zugreifen. Diese Funktion ist nur ein „Kann-Kriterium“ und ist nicht unbedingt nötig.

## **Sicherheits- und Datenschutzaspekt**

### **Benutzererkennung**

Für die Applikation selber ist keine eigentliche Benutzererkennung erforderlich. Alle Personen, die sich die Software herunterladen und installieren, können die Applikation uneingeschränkt nutzen.

Um die Software jedoch herunterzuladen und zu verwenden muss man einmalig einen Barcode als Registrierung fotografieren. So erhält man eine User Identifikationsnummer und diese wird dann automatisch in die interne Datenbank von Android gespeichert.

### **Persönliche Daten**

Es werden keine persönlichen Daten gespeichert. Wie vorgängig erwähnt, wird die eindeutige User Id des Benutzers in die Datenbank gespeichert. Diese Nummer hat jedoch keinerlei Zusammenhang mit der Person. Sie ist eine zufällig generierte Nummer.

Auch durch die Verbindung zum Internet werden keine Informationen gespeichert.

### **Gesicherter Zugriff via HTTPS**

Durch den gesicherten Zugriff via HTTP ist es zudem nahezu unmöglich, dass sich Personen unerwünscht in das System einschleusen.

### **Gespeicherte Daten**

Zurzeit ist geplant, dass nur die User ID, welcher der Benutzer bei der Registrierung auf der Programmwebseite erhält, abgespeichert wird.

Zu einem späteren Zeitpunkt wird geprüft, ob eine Historie sinnvoll wäre. So könnte der Benutzer beispielweise auf die vorhergegangenen Suchanfragen zurückgreifen und die Produkte immer wieder anzeigen lassen. Bei diesem

Szenario werden alle Barcodes in der internen Datenbank gespeichert und weiter verarbeitet.

### **Internetzugriff**

Um Informationen zu Produkten oder Dienstleistungen zu erhalten, ist es zwingend nötig, dass die Applikation auf eine Webapplikation zugreift. Dabei können Kosten des Providers entstehen.

Um die Kosten für die Benutzer so niedrig wie möglich zu halten, werden nur die nötigsten Informationen übermittelt

## **Technische Bedingungen**

Die Bachelorarbeit muss unter folgenden technischen Bedingungen realisiert werden:

### **Entwicklungsumgebung**

Da Google eine vollumfängliche Android SDK zur Verfügung stellt, welche sich auf die Entwicklungssprache Java bezieht, wird diese Programmiersprache verwendet.

### **Hardware**

Die Person, welche die Software verwenden möchte, ist auf ein Android-Smartphone angewiesen. Das Smartphone muss zudem über eine Kamera mit Videofunktion verfügen.

Um Informationen zu den Produkten anzuzeigen ist eine Internetverbindung nötig. Das Smartphone muss also so konfiguriert sein, dass man auf das Internet gelangen kann.

### **Software**

Die Applikation setzt das Betriebssystem Android voraus. Ohne dieses Betriebssystem kann die Applikation nicht gestartet und verwendet werden.

Zudem muss ein funktionsfähiger Browser installiert sein, welcher den Zugriff auf das Internet sicherstellt.

## Qualitätsbestimmungen

Auf welche Qualitätsanforderungen wird bei der Arbeit / dem Produkt besonderer Wert gelegt? Die Gewichtung wird in Stufen von 1 – 4 eingeteilt, wobei 1 die niedrigste und 4 die höchste Gewichtung trägt.

	1	2	3	4
<b>Applikation</b>				
Zuverlässigkeit				X
Fehlertoleranz				X
Genauigkeit der Barcodeerkennung				X
Anpassbarkeit durch den Benutzer	X			
Bedienbarkeit durch den Benutzer			X	
Kompatibel mit anderen Plattformen	X			
<b>Source Code</b>				
Verständlichkeit			X	
Effizienz des Source Codes				X
Ausführliche Dokumentation des Codes			X	
<b>Dokumentation</b>				
Technische Dokumentation			X	
Android Dokumentation		X		

## Bestimmungen

Dieses Pflichtenheft wird als gültig angesehen, wenn es von den oben erwähnten Partnern unterzeichnet wurde. Alle Änderungen der Funktionen dürfen nur nach Absprache stattfinden.

Verantwortlicher HES-SO Valais

Verantwortlicher RFID Center

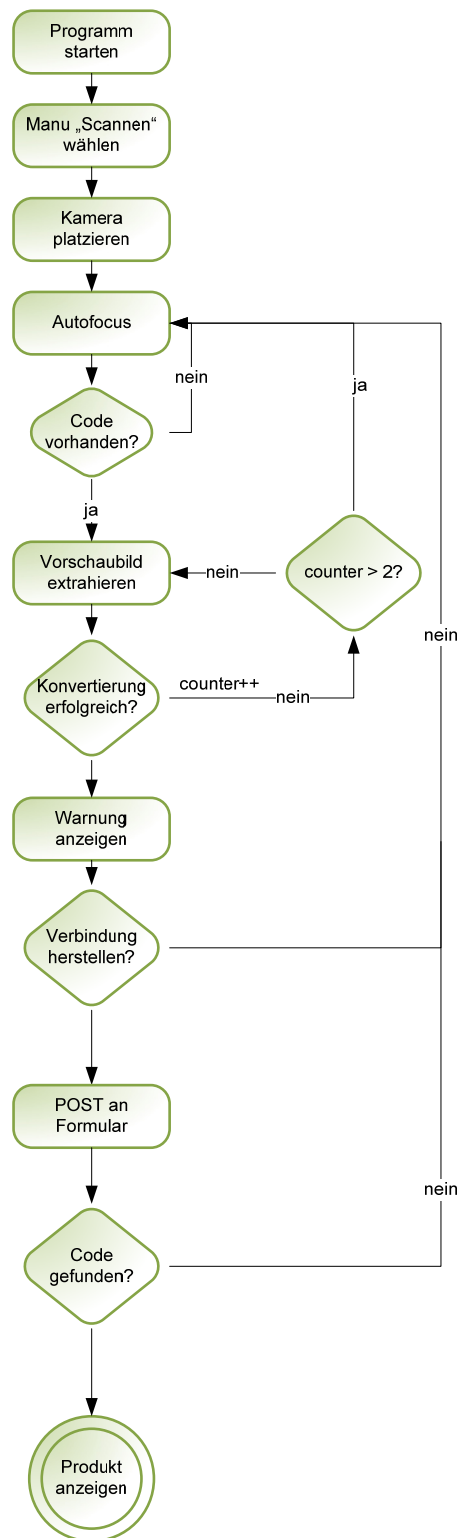
*Laurent Sciboz*

*Jean-Jérôme Sarrazin*

Student / Entwickler

*Stefan Theler*

## Anhang 3 – Bildanhänge und Planung





# Wochenplanung

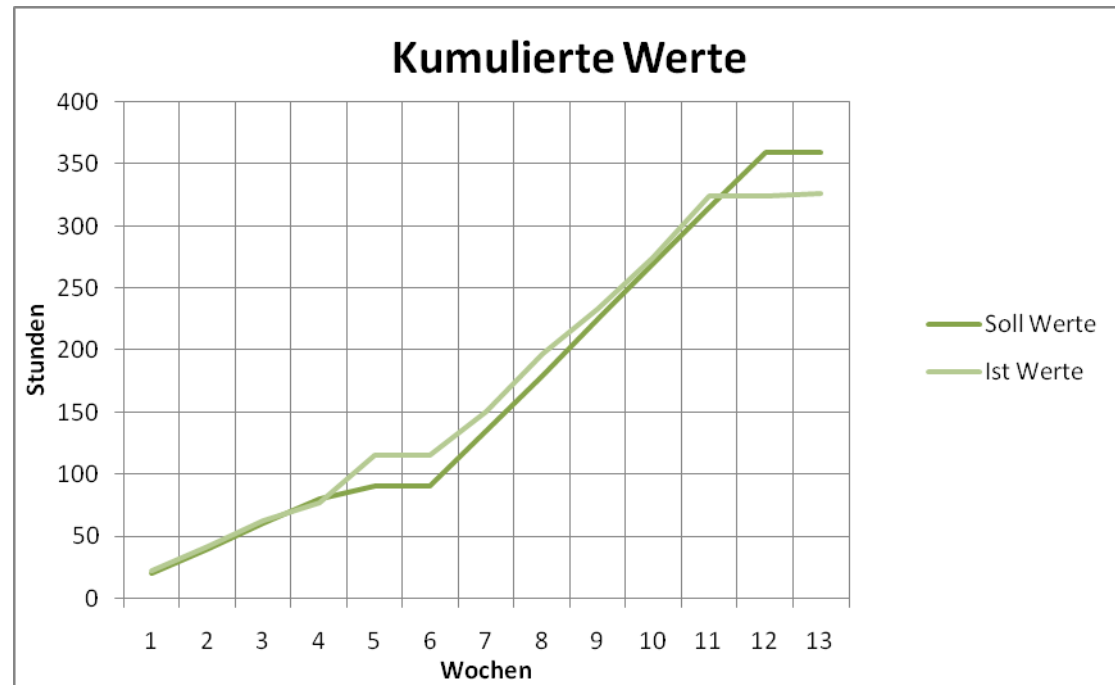
Wochenplanung	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	Total Stunden pro Thema / Bereich	
	11.05 13.05	18.05 20.05	25.05 27.05	01.06 03.06	08.06 10.06	15.06 17.06	22.06 27.06	29.06 04.07	06.07 11.07	13.07 18.07	20.07 25.07	27.07 01.08	03.08 03.08		
Vorbereitung / Themenübergabe															Total 7 h
Beginn Diplomarbeit	0													0 h	
Planung erstellen	2													2 h	
Pflichtenheft erstellen	2	3												5 h	
Einarbeit in das Thema															Total 40 h
Sammeln von Informationen (Bücher, Internet, Tutorials)	5	5												10 h	
Installation der Android SDK	1													1 h	
Einarbeit in die Android SDK / Bachelorarbeit	6	8	15											29 h	
Analyse Android Market															Total 8 h
Informationssuche Android Market				8										8 h	
Analyse Dekodierungs-Applikation des RFID-Centers															Total 10 h
Barcodeleser-Code analysieren und verstehen				6	4									10 h	
Entwicklung des Barcode Lesers															Total 160 h
Erstellung der Menüs und Funktionen							40	20						60 h	
Barcode-Handling (Erfassen, Erkennen, Verarbeiten)								20	40					60 h	
Sicherheit implementiern (HTTPS, Webserver)										20				20 h	
Speichern der Daten im Memory des PDA's										20				20 h	
Qualitätssicherung / Tests															Total 35 h
Qualitätsprüfung											20			20 h	
Test der Mobilität (verschiedene Android Handymodelle)											15			15 h	
Dokumentation															Total 79 h
Dokumentation erstellen	2	2	3	4	5		3	3	3	3	8	43		79 h	
Projektverwaltung, Meetings und Verschiedenes															Total 21 h
Projektverwaltung, Meetings und Verschiedenes	2	2	2	2	1		2	2	2	2	2	2		21 h	
Stunden Berechnung:	20	20	20	20	10		45	45	45	45	45	45	0	Total 360 h	
Soll Stunden	20	20	20	20	10		45	45	45	45	45	45	0	360 h	
Differenz	0	0	0	0	0		0	0	0	0	0	0	0	0 h	

# Effektive Stunden

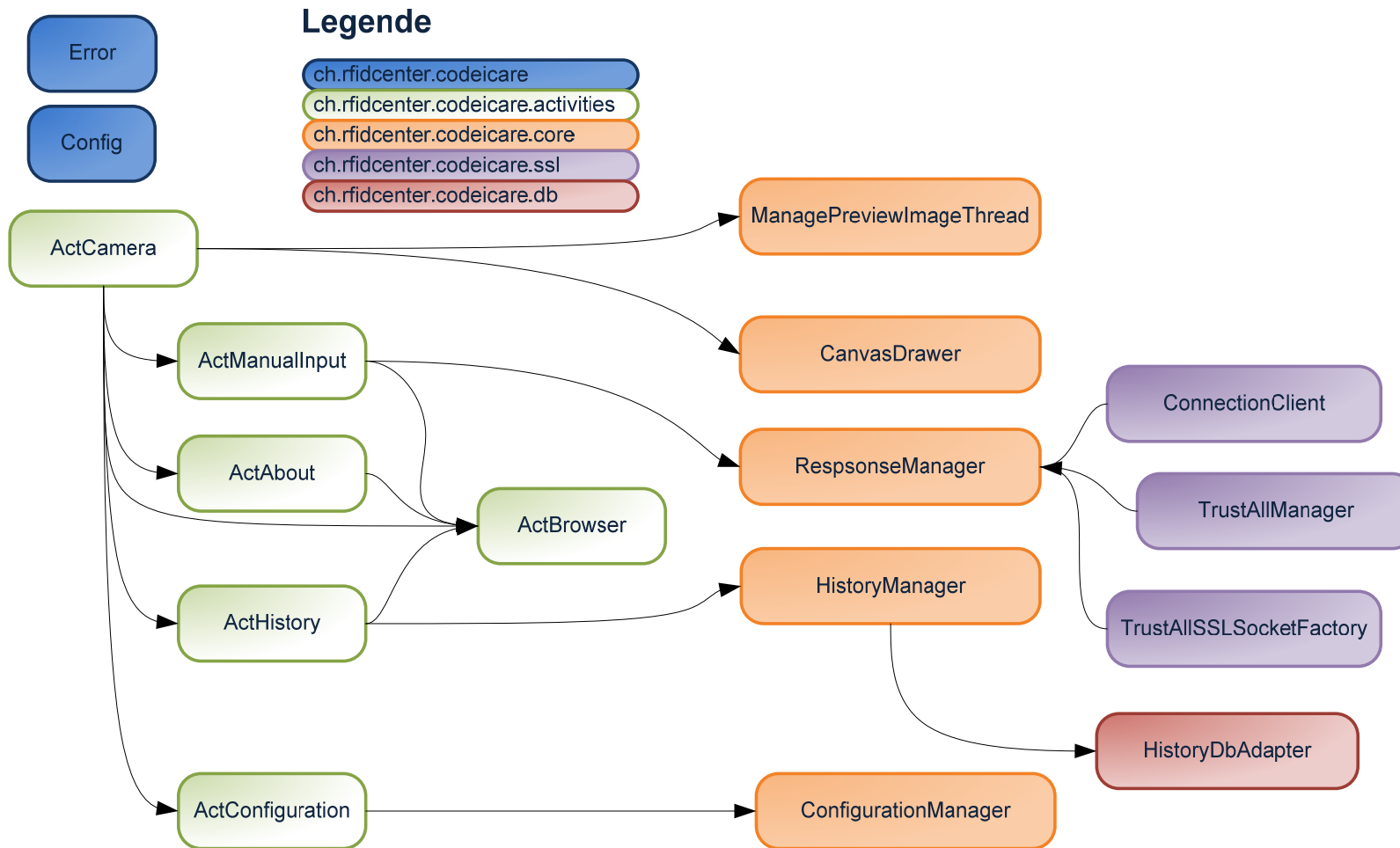
	W1 11.05 13.05	W2 18.05 20.05	W3 25.05 27.05	W4 01.06 03.06	W5 08.06 10.06	W6 15.06 17.06	W7 22.06 27.06	W8 29.06 04.07	W9 06.07 11.07	W10 13.07 18.07	W11 20.07 25.07	W12 27.07 01.08	W13 03.08 03.08	Total Stunden pro Thema / Bereich
<b>Vorbereitung / Themenübergabe</b>														<b>Total 7 h</b>
Beginn Diplomarbeit	0													0 h
Planung erstellen	2													2 h
Pflichtenheft erstellen		5												5 h
<b>Einarbeit in das Thema</b>														<b>Total 41 h</b>
Sammeln von Informationen (Bücher, Internet, Tutorials)	6	3												9 h
Installation der Android SDK	2													2 h
Einarbeit in die Android SDK / Bachelorarbeit	8	8	14											30 h
<b>Analyse Android Market</b>														<b>Total 6 h</b>
Informationssuche Android Market				6										6 h
<b>Analyse Dekodierungs-Applikation des RFID-Centers</b>														<b>Total 3 h</b>
Barcodeleser-Code analysieren und verstehen				1	2									3 h
<b>Entwicklung des Barcode Lesers</b>														<b>Total 183 h</b>
Erstellung der Menüs und Funktionen					24		2	8	15	10	5	4		68 h
Barcode-Handling (Erfassen, Erkennen, Verarbeiten)					10		30	25	15	4	15			99 h
Sicherheit implementieren (HTTPS, Webserver)								4		4				8 h
Speichern der Daten im Memory des PDA's								8						8 h
<b>Qualitätssicherung / Tests</b>														<b>Total 5 h</b>
Qualitätsprüfung												5		5 h
Test der Mobilität (verschiedene Android Handymodelle)														0 h
<b>Dokumentation</b>														<b>Total 100 h</b>
Dokumentation erstellen	2	2	4	6					4	24	20	38		100 h
<b>Projektverwaltung, Meetings und Verschiedenes</b>														<b>Total 24 h</b>
Projektverwaltung, Meetings und Verschiedenes	2	2	2	2	2		3	2	2	3	1	3		24 h
														0 h
<b>Stunden Berechnung:</b>	22	20	20	15	38	0	35	47	36	45	41	50	2	<b>Total 371 h</b>
<b>Soll Stunden</b>	20	20	20	20	10	0	45	45	45	45	45	45	0	360 h
<b>Differenz</b>	2	0	0	-5	28	0	-10	2	-9	0	-4	5	2	11 h

## Anhang 6 – Statistischer Überblick Planung – Effektive Zeit

	Soll Stunden	Kumuliert	Ist Stunden	Kumuliert
Woche 1	20	20	22	22
Woche 2	20	40	20	42
Woche 3	20	60	20	62
Woche 4	20	80	15	77
Woche 5	10	90	38	115
Woche 6	0	90	0	115
Woche 7	45	135	35	150
Woche 8	45	180	47	197
Woche 9	45	225	36	233
Woche 10	45	270	41	274
Woche 11	45	315	50	324
Woche 12	45	360	0	324
Woche 13	0	360	2	326



## Anhang 7 - Klassenhierarchie



## Index

### A

- adb 29
- Aktivierung 39
- AlertBox 42
- Android Runtime 15
- Application Framework 16
- ARGB 50, 51

### B

- Barcodes 31

### D

- Dalvik Virtual Machine viii, 11, 16
- DDMS 26

### E

- EAN 32

### F

- Fehlerhandling 42

### G

- GEPiR 35
- GS1 35

### H

- Handler 49, 52, 53
- History 17, 37, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 81, 104, 105, 106, 111, 113
- History Stack 17

### I

- installAPK 14

### J

- JNI viii, 43, 57, 58, 102

### K

- Klassenhierarchie 36

### L

- Linux Kernel 15
- Live-Android 11
- LogCat 27

### M

- Manifest 21
- Mehrsprachigkeit 65

### N

- NDK 58

### O

- Open Handset Alliance 12, 13

### P

- POST 55
- Preferences 65

### Q

- QR-Codes 32
- Querformat 22

### R

- Registrierung 39
- Resource Manager 17
- Response Manager 52
- RFID 31

### S

- Shared Library* 51, 57, 60, 61, 64
- SharedPreferences 67

### T

- Threads 27
- Timeline 13
- Toast 42
- Traceview 29

### V

- Versioning* 22
- Views 20

### W

Web-View- 57

### Y

YUV 420 49, 51