DEUTSCHE TELEKOM: HOCHSCHULE FÜR TELEKOMMUNIKATION LEIPZIG

Dokumentation zum Modul Mobile Applikationen

Thema **NFC Visitenkarten App**

vorgelegt von
Oliver Friedrich
Marko Klepatz
Klaus Steinhauer

Betreuer
Prof. Ulf Schemmert

Source Code https://github.com/NFCdroid/nfcapp

Google Play Store https://play.google.com/store/apps/details?id=com.ag.mk.nfccardreadwrite

Inhaltsverzeichnis

1 Einleitung

Im Rahmen unseres Studiums der Wirtschaftsinformatik haben wir uns im 5. Semester für die Profilierung Mobile Applikationen entschieden. Hauptbestandteil und auch prüfungsrelevant für das Modul ist das eigenständige Planen und Implementieren einer Android App.
Nach einer kurzen Einführung in die Android Programmierung durch Herrn Professor Schemmert lag es an den Studierenden sich für eines der vorgeschlagenen Projekte oder den Entwurf eines eigenen Projektes zu entscheiden.
Wir haben uns dazu entschieden, eine von uns vorgeschlagene NFC[1] App zu programmieren. Diese wurde nach einer kurzen Rücksprache mit Herrn Professor Schemmert genehmigt.
Wir werden das gesamte Projekt gemäß unserer in vorherigen Semestern erlernten Methoden planen und mit den erlernten Mitteln des Moduls Software-Engineering an die Implementierung herangehen.

Im Folgenden soll unsere Herangehensweise, aufgetretene Schwierigkeiten und weitere Informationen aufgezeigt werden.

1.1 Motivation

Auf der Suche nach einer App mit praktischem Mehrwert und interessantem Inhalt haben wir uns für eine NFC App entschieden.
Nachdem die anfängliche Idee eines Klonens der HfTL Card, um ein Handy als Zugangsmedium zu den Räumen nutzen zu können, sich bald als unrealistisch herausstellte, haben wir uns entschieden eine VisitenkartenApp für NFC Karten zu schreiben.
Aufgrund der Preisentwicklung und umweltschonenden Eigenschaften von elektronischen Visitenkarten scheint ein "baldiges" Ersetzen der klassischen Visitenkarte aus Papier realistisch. Zudem werden Kontakte heute in der Regel digital gespeichert und verwaltet was ein analoges Austauschen von Daten redundant macht.

1.2 Ziel

Bis zur Abgabe des Projektes zum Ende des WS 2015/16 soll eine App programmiert werden die Kontakte aus Android oder von einer Eingabemaske einlesen und auf eine NFC Karte speichern, sowie von der Karte lesen und in den Android Kontakten speichern kann.
Zusätzlich sollen die Kontaktinformationen direkt zwischen zwei NFC fähigen Geräten kontaktlos ausgetauscht werden können. Dafür soll eine übersichtliche Oberfläche mit maximal vier Buttons erstellt werden und eine Hilfe für den Nutzer zur Verfügung stehen.

[1] Near Field Communication: auf RFID basierender Übertragungsstandard

1.3 Anforderungen

Ausgehend von unserer Zielformulierung und der Motivation ergeben sich einige funktionale und nicht funktionale Anforderungen. Zu den Funktionalen gehören das Lesen und Schreiben von Kontakten in Android, sowie der NFC Karte. Zudem soll es möglich sein die Daten Kontaktlos zu anderen Android Geräten übertragen zu können.

Um die Akzeptanz der App und damit auch deren Verbreitung zu unterstützen ergeben sich Anforderungen an die optische Gestaltung der App. Sie sollte möglichst übersichtlich, intuitiv und auch adaptiv an die Gerätegröße gestaltet sein. Zusätzlich ist es uns wichtig, eine Dokumentation für den Nutzer zu erstellen die den nötigen Support auf ein Minimum reduziert und eine frustfreie Verwendung der App ermöglicht.

2 Technische Einordnung

NFC, ist eine auf RFID[1] basierende Technologie zur kontaktlosen Datenübertragung.
Hierbei können die Daten auch "passiv" verfügbar sein, d.h. der NFC-Chip beinhaltet keine eigene Stromversorgung und wird erst durch ein "aktives" Lesegerät ausgelesen.
Die Übertragung erfolgt per Funk innerhalb eines Frequenzbereichs 13,56 MHz, hierbei ist eine maximale Datenübertragungsrate von 424 kBit/s möglich, bei einer Reichweite von maximal 10 cm. Haupteinsatzbereiche sind kontaktloses Bezahlen und Zugangskontrolle, jeweils unter Einsatz eines verschlüsselten Datenbereiches, auf den der Zugriff mittels Challenge-Response-Verfahren erfolgt.
Als Datenaustauschformat kommt meist NDEF[2] zum Einsatz. Weitere Informationen zur Spezifikation von NFC-Karten können der ISO 7816 entnommen werden.

Seit 2008 gibt es erste NFC-fähige Mobiltelefone. Ein großer Teil der aktuellen Android-Smartphones verfügt über einen integrierten NFC-Chip.
Dieser beinhaltet bereits eigene Verschlüsselungscodes, stellt seine Funktionalität aber auch per Android API zur Verfügung. Zusätzlich zu der zuvor beschriebenen aktiv-passiv-Kopplung wurde mit "Android Beam" auch eine aktiv-aktiv-Kopplung zwischen zwei Android-Geräten eingeführt.

Laut offizieller Android Entwicklungsdokumentation werden drei verschiedene NFC-Modi bereitgestellt:

- Lesen/Schreiben von NDEF-Daten eines NFC-Chips (seit Android 2.3)
- Beamen von Daten zu anderen Androidgeräten mittels Android Beam (seit Android 4.0)
- Emulation eines NFC-Chips ("Host-based Card Emulation", seit Android 4.4)

Im Rahmen dieser Projektarbeit war die Implementierung aller drei genannten NFC-Technologien geplant, jedoch konnten wir nur das Lesen und Beamen umsetzen, da es bei der NFC-Chip-Emulation zu unerwarteten Problemen kam (siehe Problembetrachtung).
Zu Testzwecken wurden wiederbeschreibbare NFC-Karten mit dem Chip NTAG 216 und 888 Byte Speicher beschafft. Als Test- und Entwicklungsgeräte dienten zwei Samsung Galaxy S5 (Android 5.1 und 6.0) sowie ein Samsung Galaxy S3 (Android 4.3).
NFCTagger ist eine App um Visitenkarten im VCard Format auf NFC Karten zu speichern, auszulesen und kontaktlos auszutauschen.

[1] radio-frequency identification: Technologie zur kontaktlosen Datenübertragung mittels elektromagnetischer Wellen
[2] NFC Data Exchange Format, Standard-Datenaustauschformat zwischen NFC-Chips

3 Anwenderdokumentation des NFCTaggers:

3.1 Unterstützte NFC Karten

Unterstützt werden alle Karten die das NDEF Format unterstützen. Sollte eine Karte nicht beschreibbar sein kontaktieren Sie uns bitte unter marko.klepatz@hotmail.de

3.2 Kontakte auf einer Karte Speichern

Sie können Kontakte entweder aus dem Android Kontaktbuch importieren oder manuell erstellen und auf einer NFC Karte speichern. Wählen Sie "Kontakt Import" um Daten aus dem Kontaktbuch zu importieren oder "V-Card Erstellen" um die Daten manuell einzugeben.

3.2.0.1 Kontakt Import

Wählen Sie einen beliebigen Kontakt zum Import. Nach erfolgtem Import können Sie die Daten mit "V-Card Erstellen" noch abändern und anschließend auf die Karte speichern.

3.2.0.2 V-Card Erstellen

Hier können Sie eigene Daten eingeben oder die vom Import eingefügten Daten bearbeiten. Legen Sie ihre NFC Karte auf die Rückseite des Gerätes und wählen Sie "Karte Beschreiben". Alternativ können Sie die Daten auch übernehmen und per Android Beam an ein anderes Gerät senden.

3.3 Kontakte von einer Karte lesen

Legen Sie die Karte einfach auf die Rückseite ihres Gerätes auf um die gespeicherten Daten angezeigt zu bekommen.

3.3.1 Ausgelesenen Kontakt in Android Kontaktbuch Speichern

Sie können den gelesenen Kontakt als neuen Kontakt speichern oder einen vorhandenen Kontakt ergänzen. Wählen Sie einen vorhandenen Kontakt oder das Plus um einen neuen zu erstellen.

3.3.2 Android Beam

Nach dem Erstellen, Auslesen oder Importieren eines Kontaktes können Sie diesen mit Android Beam an ein anderes Android Gerät übertragen. Halten Sie die Geräte mit dem Rücken aneinander und wählen Sie im Hauptdialog "Android Beam". Bitte beachten Sie dass es zu Problemen kommen kann wenn die Geräte durch Hüllen geschützt sind.

3.4 Einstellungen

3.4.1 Sprachausgabe

Liest Hinweistexte zur Interaktion mit der App vor. Nutzt Google Text-to-Speech

3.4.2 Vibration

De-/Aktiviert die Vibration bei Erkennung einer neuen Karte. Ausgehend von unserer Zielformulierung und der Motivation ergeben sich einige funktionale und nicht funktionale Anforderungen. Zu den Funktionalen gehören das Lesen und Schreiben von Kontakten in Android, sowie der NFC Karte. Zudem soll es möglich sein die Daten Kontaktlos zu anderen Android Geräten übertragen zu können.

4 Planung

4.1 Meilensteinplan

Wegen des fest vorgegebenen Semesterablaufes ist unsere Zeitplanung bereits weitestgehend festgelegt. Wie im Anhang [link] ersichtlich ist haben wir einen Projektabschluss zum 21.01.↩ 2016 geplant. Um den Fortschritt im Auge behalten zu können haben wir folgende Meilensteine geplant:

ID	Meilenstein	Datum
M0	Projektstart	22.10.2015
M1	Vorstellung Projektziel	12.11.2015
M2	Dokumentation, Teil 1 fertiggestellt	29.11.2015
M3	Prototyp erstellt	02.12.2015
M4	App getestet + stabilisiert	16.12.2015
M5	Projektabschlussbericht erstellt	15.01.2016
M6	Projektabschluss	21.01.2016

4.2 Projektplan

Zur besseren Aufgabenverteilung wurde zu jeder Aufgabe ein Verantwortlicher bestimmt, der sich um die Einhaltung der Fristen und Anforderungen dieser Aufgabe zu kümmern hatte. Dabei wurde versucht, die Aufgaben möglichst gleichmäßig zu verteilen.

ID	Vorgang	Anfang	Ende	Dauer	Verant-wortlich	Gliederungs↩ Nr.
0	**Projekt-start**	22.10.15	22.10.15	0	Marko Klepatz	1
7	Ideenfin-dung	22.10.15	28.10.15	5	Klaus Steinhauer	2
8	Themen-auswahl	29.10.15	04.11.15	5	Oliver Friedrich	3
9	Anforde-rungsana-lyse und Projektpla-nung	05.11.15	11.11.15	5	Klaus Steinhauer	4

16	**Vorstellung Projektziel**	12.11.15	12.11.15	0	Marko Klepatz	5
15	Dokumentation Use Cases und Architektur	12.11.15	18.11.15	5	Oliver Friedrich	6
19	**Dokumentation, Teil 1 fertiggestellt**	19.11.15	19.11.15	0	Oliver Friedrich	7
20	Erstellung eines Prototypen	19.11.15	02.12.15	10	Klaus Steinhauer	8
22	Basisapp und Design	19.11.15	02.12.15	10	Marko Klepatz	8.1
25	Kartenemulation	19.11.15	02.12.15	10	Oliver Friedrich	8.2
27	vCard-↩ Anbindung	19.11.15	02.12.15	10	Klaus Steinhauer	8.3
321	**Prototyp erstellt**	02.12.15	02.12.15	0	Marko Klepatz	9
267	Testen, Bugfixing	03.12.15	16.12.15	10	Oliver Friedrich	10
323	**App getestet + stabilisiert**	22.10.15	22.10.15	0	Klaus Steinhauer	11
303	Projektabschlussbericht erstellen	17.12.15	14.01.16	21	Oliver Friedrich	12
300	Anwenderdokumentation erstellen	17.12.15	28.12.15	8	Marko Klepatz	12.1
318	Klassendokumentation generieren	29.12.15	06.01.16	7	Oliver Friedrich	12.2

319	Fehler- und Problembetrachtung	07.01.16	14.01.16	6	Klaus Steinhauer	12.3
308	**Projektabschlussbericht erstellt**	15.01.16	15.01.16	0	Marko Klepatz	13
310	Projektpraesentation erstellen	15.01.16	21.01.16	5	Klaus Steinhauer	14
314	Projektabschluss vorbereiten	15.01.16	21.01.16	5	Oliver Friedrich	15
312	**Projektabschluss**	21.01.16	21.01.16	0	Marko Klepatz	16

Trotz einiger unerwarteter Probleme konnte der aufgestellte Plan zum Großteil eingehalten werden.

5 Software Engineering

5.1 Usecase Diagramm

Zu Beginn des Projektes ist es wichtig, Klarheit über die benötigten Funktionalitäten zu schaffen. State-of-the-art ist hier das Erstellen eines Usecase-Diagramms um die für den Nutzer möglichen Anwendungsfälle zu erfassen und die Programmierung darauf auszurichten. Hierbei ist insbesondere die Interaktion zwischen den Nutzern hervorzuheben. Beide nutzen die selbe App und sind in der Lage, Inhalte über Android Beam auszutauschen.

5.2 Activity Diagramm

Zu Beginn der Implementierung ist es wichtig, sich ausgehend vom Usecase Diagramm Klarheit über den logischen Aufbau der App zu machen, hier ist ein Aktivitätsdiagramm bestens geeignet. Hier werden auf übersichtliche Weise alle möglichen Bedienpfade der App aufgezeigt. Auch die innere Struktur der App ist entsprechend strukturiert.

5.3 Klassendiagramm

Nachdem die App auf die Bedienpfade abgestimmt wurde, ist es gute Praxis, die Funktionen logisch in verschiedene Klassen zu teilen. Anfangs wurde der Großteil in der Main Activity implementiert. Im Rahmen mehrerer Refactorings wurden immer mehr Funktionen in Klassen ausgelagert und diese in der Main Activity referenziert. Das zum Ende entstandene Klassendiagramm versucht die Abhängigkeit der Klassen untereinander aufzuzeigen.

Abbildung 5.1 **Usecasediagramm**

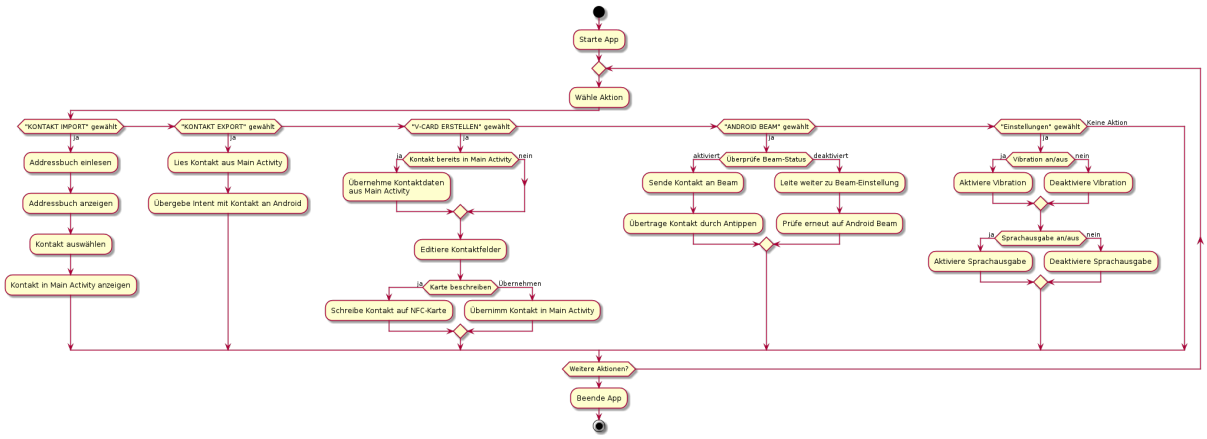

Abbildung 5.2 **Aktivitätsdiagramm**

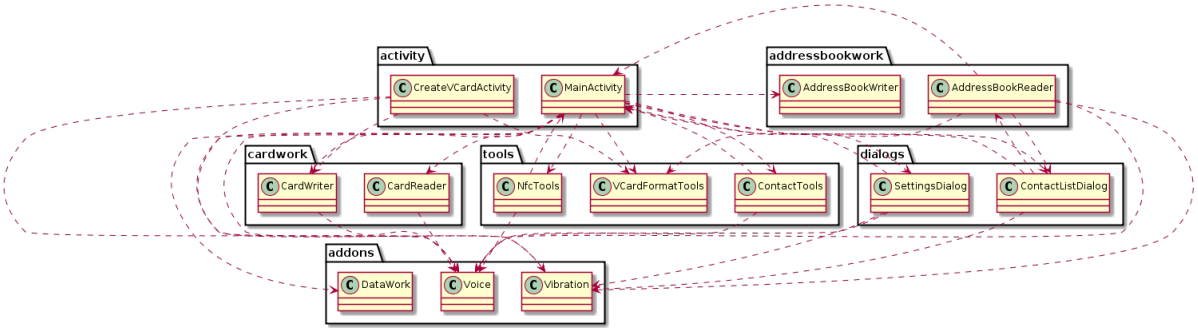

Abbildung 5.3 **Klassendiagramm**

6 Implementierung

6.1 Vorgehensweise

6.1.1 Extreme Programming

Das Team arbeitete mit den agilen Methoden des „Extreme Programming“ (XP). Die Entscheidung zur Anwendung dieser Methode ist hauptsächlich durch den Projekttypen und die zu Projektbeginn zur Verfügung stehenden Informationen bedingt.
Da es sich um ein prototypisches Programmierprojekt handelt, war der Aufwand zur Implementierung einiger Funktionalitäten schwer abzuschätzen. Weiterhin war unser "Kunde", Professor Schemmert, sehr gut verfügbar und auch in seinen Anforderungen flexibel.
Voraussetzung für die erfolgreiche Durchführung des XP waren der Einsatz einer verteilten Versionsverwaltung (siehe nächster Abschnitt), kontinuierliches Testen auf unterschiedlichen Geräten sowie eine teilautomatisch generierte Dokumentation.

Natürlich konnte nicht jede Technik genau nach diesem Modell angewandt werden, so waren beispielsweise „Stand Up Meetings“ nicht täglich sondern nur wöchentlich möglich.
Das Durchführen von „Unit Tests“ war ebenfalls zeitlich nicht realisierbar, zukünftig könnten diese aber durch den Einsatz von "Continuous Integration[1]" abgedeckt werden.
Jedoch hat die XP-Technik des Pair-Programming immer wieder geholfen, Fehler, Lösungen und Code-Optimierungen schnell zu finden und zu realisieren.
Auch der gegenseitige bedingte Wissenszuwachs, der in einem Hochschulprojekt mit im Vordergrund stehen sollte, war dadurch sehr groß.

6.1.2 programmiertechnisches Vorgehen

Das Team ist programmiertechnisch folgendermaßen vorgegangen: Es wurde zunächst überlegt welche Klassen und Methoden implementiert werden müssen, nachdem diese realisiert wurden und im späteren Verlauf Abhängigkeiten entstanden, wurden diese agil hinzugefügt.
Sollte eine Klasse oder Methode zu groß geworden sein, wurden diese durch Refactoring[2] in dafür sinnvolle kleinere Klassen und Methoden ausgelagert. Wenn neue Elemente implementiert wurden, wurde danach oder dabei immer direkt getestet, somit konnten wir große Fehlprogrammierungen effektiv vermeiden.

Am Ende des Projekts wurde noch ein großes Refactoring durchgeführt um dem Code eine gute Leserlichkeit zu verleihen. Dabei wurden unter anderem Klassen und Methoden ausgelagert um die Architektur zu verbessern und intuitiver zu gestalten.
Des Weiteren wurden Tests durchgeführt um letzte Logikfehler zu beseitigen und die Speichereffizienz so wie die Performance zu steigern. Dabei bedienten wir uns unter anderem der Tools „Inspect Code“ und dem Ressourcenmonitor, welche in Android Studio integriert sind.

[1] kontinuierliches Erstellen (Kompilieren) der Anwendung zur Steigerung der Softwarequalität
[2] Begriff aus dem Software Engineering, Restrukturierung des Quellcodes zur besseren Lesbarkeit

6.2 Technische Besonderheiten

Im Verlauf der Entwicklung gab es einige technische Besonderheiten zu beachten. Einen guten Überblick dazu bietet ein Blick in die **AndroidManifest.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ag.mk.nfccardreadwrite">
```

Hier werden alle benötigten Berechtigungen der App aufgelistet. Einige Berechtigungen haben sich erst im Verlauf der Entwicklung ergeben, beispielsweise CALL_PHONE und VIBRATE.

```
    <uses-permission android:name="android.permission.NFC" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.VIBRATE" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
```

Hier erhält die App Zugriff auf den NFC-Chip.

```
    <uses-feature
        android:name="android.hardware.nfc"
        android:required="false" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/tagger_logo"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".activity.MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.nfc.action.NDEF_DISCOVERED" />

                <category android:name="android.intent.category.DEFAULT" />
```

Zum Schreiben von NDEF-Daten müssen die Daten auf dem Chip mit einem speziellen MIME-Typ gespeichert werden. So können Kontaktdaten mit dem Standard-vCard MIME-Typ geschrieben werden und das Android System verwaltet automatisch die damit registrierten Anwendungen, so würde sich in diesem Fall jede mit vCard registrierte App öffnen lassen, z.B. die Android Kontakte-App.

Wir haben uns bewusst gegen einen generischen MIME-Typ entschieden und stattdessen unseren eigenen gebaut: application/vnd.com.ag.mk.nfccardreadwrite.beam. Ein praktischer Nebeneffekt ist die auf einigen Geräten existierende Verknüpfung unbekannter MIME-Typen mit dem Google PlayStore. Hierbei ist es möglich, die App nur durch Auflegen auf eine NFC-Karte aus dem PlayStore herunterzuladen, da nur diese den von uns defninierten MIME-Typ unterstützt.

```
                <data android:mimeType="application/vnd.com.ag.mk.nfccardreadwrite.beam"
            </intent-filter>
            <intent-filter>
                <action android:name="android.nfc.action.TECH_DISCOVERED" />
                <action android:name="android.nfc.action.TAG_DISCOVERED" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>

            <meta-data
                android:name="android.nfc.action.TECH_DISCOVERED"
                android:resource="@xml/tech" />
        </activity>
        <activity android:name=".activity.CreateVCardActivity" >
            <intent-filter>
                <action android:name="android.intent.action.SEND" />

                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
```

Eine weitere Besonderheit ist das systemweite Eintragen der App als Empfänger für geteilte Kontakte. Dazu wird die App auf den zu empfangenden MIME-Typ, hier für das Standard vCard-Format, registriert und kann nun den vCard-senden-Intent des Systems empfangen.

```
            <intent-filter>
                <action android:name="android.intent.action.SEND" />

                <category android:name="android.intent.category.DEFAULT" />

                <data android:mimeType="text/x-vcard" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />

    </application>

</manifest>
```

Natürlich wäre es ohne weiteres möglich gewesen, unsere App auf den generischen vCard-↩ MIME-Typ zu registrieren, intern handelt es sich um die gleiche Datenstruktur. Allerdings haben wir uns an dieser Stelle für den "exklusiveren" Weg entschieden.

6.3 Dokumentation

Für die Dokumentation unserer Arbeit haben wir verschiedene Teilautomatische Open Source Lösungen verwendet. Zum Erstellen der Java Dokumentation haben wir Doxygen genutzt. Zum Erstellen der UML Diagramme plantuml, welches die Möglichkeit bietet, UML Diagramme mittels einfacher Textbeschreibung automatisiert zu erstellen. Beide Tools bieten unter anderem LaTeX Dokumente als Ausgabeformat, welche wir als Basis für unsere Dokumentation verwendeten.

6.4 Problembetrachtung

Doppelter Intent

Der Fehler, welcher das größte Fehlverhalten verursachte und leider auch nicht durch das Analysieren von Exceptions zu beseitigen war, weil er zum Teil logischer Natur entsprang, war jener einer ungenutzten implementierten Technologie im Intent-Filter „IsoDep.class.getName()“.
Er wies folgendes Verhalten auf: Wann immer ein NFC Medium an das Gerät gehalten, bzw. ein Beam Vorgang gestartet wurde, startete die App automatisch neu.
Erst durch das Entfernen der Funktion aus dem Intent-Filter, konnte dieser Fehler behoben werden.

Zugriff auf das Secure Element

Zur Kommunikation mit Bezahlterminals und Zugangssystemen verfügen Android Smartphones über ein Secure Element innerhalb des NFC-Chips. In diesem Secure Element sind IDs und Schlüssel einiger sicherheitskritischer Anwendungen gespeichert, weshalb der direkte Zugriff darauf (via API o.ä.) nicht möglich ist.
Es kann nur die Route zum und vom Secure Element angesprochen werden, d.h. die Daten werden z.B. von der App bereitgestellt und das Secure Element stellt die Sicherheitsfunktion bereit.

Kartenemulation

Zur Emulation von Karten kann auch das Secure Element des Gerätes verwendet werden, hierbei ist jedoch zwischen den unterschiedlichen NFC-Chips zu unterscheiden, da diese nicht jeden Kartentyp emulieren können.
Zur Unterscheidung der NFC-Lesegeräte werden diese intern mit einer Typ-ID versehen und im Android-System entsprechend geroutet. Dieses Routing verhindert auch den weiteren Einsatz der Kartenemulation.
Das Hauptproblem hierbei ist das statische Routing der AID[3] auf spezielle Apps. Wenn es zu der AID des Lesegerätes keine im System registrierte App gibt, dann kommt der initiale Handshake zwischen emulierter Karte und Lesegerät nicht zustande. Der Großteil der Hersteller verwendet eigene AIDs und stellt diese meist nicht zur Verfügung, weiterhin werden diese AIDs vom Android-System zwar registriert, aber sind nicht ohne erheblichen Aufwand (Änderung der Berechtigungsdatei mit root-Rechten...) auszulesen.
Folglich wäre es notwendig gewesen, die App auf mehrere AIDs zu registrieren. Für die anfängliche Idee, die HfTL-Zugangskarte zu emulieren, hätte selbst die Kenntnis der AID nicht ausgereicht, da diese ein Secure Element beinhaltet und dieses von der HCE[4] nicht abgedeckt werden kann.

[3] Application ID: eindeutige ID des NFC-Readers
[4] Host based card emulation: Emulation einer NFC-Karte durch das Gerät

Ein weiteres Problem hinsichtlich der HCE ist auch eine mangelnde Testumgebung, für realistische Tests wäre die Anschaffung eines NFC-Lesegerätes unumgänglich gewesen, da der in Android integrierte NFC-Lesemodus zu generisch ist und sich nicht auf spezielle AIDs beschränkt.
Angenommen, die AID steht zur Verfügung, dann kann eine Verbindung der emulierten Karte zum Lesegerät aufgebaut werden und der Datentransfer beginnt. Im Anschluss ist ein weiteres Problem zu erwarten, da das Lesegerät eine APDU[5] an die emulierte Karte schickt und von dieser eine valide APDU Response erwartet.
An dieser Stelle wäre erneut die Manipulation einer validen Verbindung nötig gewesen um die korrekte APDU Response auszulesen. In Anbetracht dieser technischen Einschränkungen haben wir uns gegen die Implementierung der Kartenemulation entschieden.

6.5 Entwicklungsmodell

Zur Umsetzung der Entwicklung wurde von Anfang an auf das verteilte Versionskontrollsystem git gesetzt. Ungetestete Funktionen wurden in Branches entwickelt und erst bei voller Funktion in den dadurch stabil gehaltenen Master integriert ("merged").
Falls dieser Master doch einen fehlerhaften Zustand aufwies, so ließ sich dieser problemlos aufgrund der Commit-Beschreibungen auf einen funktionierenden Stand zurücksetzen. Um dies zu vermeiden erfolgte das Hochladen ("Commit & Push") in den Master erst nach vorherigem Testen der App.

Als technische Umsetzung wurde der git-Hosting-Dienst GitHub genutzt. Dort wurde eine Organisation erstellt die wiederum mehrere git-Verzeichnisse ("Repositories") beinhaltet.
So wurden von Anfang an die App und die Dokumentation in unterschiedlichen Verzeichnissen versioniert. Weiterhin wurde die bei GitHub integrierte Ticketverwaltung zur Projektsteuerung und Behebung von Fehlern eingesetzt.
Jedes Fehlerticket (Tag: "bug") enthält eine Beschreibung des Fehlers und kann einem Bearbeiter zugewiesen werden. Zusätzlich wurden noch Verbesserungsvorschläge als Ticket erfasst (Tag: "enhancement") und die Tickets verschiedenen Meilensteinen zugeordnet um eine bessere Terminierung zu erreichen.

[5] Application Protocol Data Unit: Anwendungseinheit innerhalb der NFC-Kommunikation, standardisiert nach ISO 7816

7 Open Source und Lizenz

Open Source

Die Ersteller der App arbeiten alle mit Linux und unterstützen dessen Philosophie der freien Weitergabe von (Quell)offener Software. Da der Quelltext der App teil der Bewertungsgrundlage ist und wir die Plattform Github verwenden war Open Source von Anfang an Teil des Entwicklungskonzeptes. Das erleichtert nicht nur die kooperative Arbeit an der App sondern bietet auch anderen Menschen die Möglichkeit den von uns erstellten Quellcode zum Lernen zu nutzen und weiterzuentwickeln.

Lizenz

Wir haben uns für die GPLv2 (GNU General Public License) entschieden, welche die Nutzung, Bearbeitung und Weitergabe des von uns erstellen Quellcodes regelt. Die GPLv2 existiert bereits seit 1991 und ist einer der am weitesten verbreiteten Softwarelizenzen für freie Software. Sie gestattet es dem Lizenznehmer die Software frei zu verwenden, bearbeiten und weiterzuverbreiten. Sie verpflichtet den Lizenznehmer dabei die Software nur mit dem selbigen Lizenzmodell weiterzugeben was eine kommerzielle Nutzung unseres Quellcodes weitestgehend ausschließt.

8 Arbeitsteilung

Programmierung

Oliver Friedrich: Beam Implementierung
Marko Klepatz: Main Activity, Helper Klassen, Refactoring, Bug-Fixing
Klaus Steinhauer: Android Kontakt Handeling

Dokumentation

Oliver Friedrich: UML, Planung, Implementierung
Marko Klepatz: Problemstellung, Quelltext Doku
Klaus Steinhauer: Ziel, Einleitung, Anforderungen, Lizenz, Fazit

9 Fazit und Ausblick

Im Laufe des Semesters konnten wir einen guten Einblick in die Android Programmierung gewinnen. Unsere Aufgabenstellung hat weite Teilbereiche der Android API abgedeckt und bietet damit eine gute Grundlage um tiefer in die Materie einzudringen.
Im Laufe des Semesters konnten wir Projektplanungs- und Software-Engineering Kenntnisse aus den vergangenen Semestern praktisch anwenden und damit den planungsgemäßen Ablauf des Projektes sicherstellen.
Trotz unterschiedlicher Vorkenntnisse in der Android Programmierung konnten wir uns alle gut in die Programmierung einbringen und jeder einzelne einen Teil beitragen.

9.1 Ausblick

Die App ließe sich auf verschiedene Weisen erweitern. Das Unterstützen von Kontaktbildern wäre wünschenswert, sofern der Speicherplatz der Karte dies zulässt, und auch andere frei wählbare vCard Felder wären denkbar.
Sinvoll wäre zudem weitere Kartentypen zu unterstützen und die Kartensimulation in die Praxis umzusetzen.

10 Anhang

10.1 Abkürzungsverzeichnis

HCE Host based card emulation: Emulation einer NFC-Karte durch das Gerät 16

APDU Application Protocol Data Unit: Anwendungseinheit innerhalb der NFC-Kommunikation, standardisiert nach ISO 7816 17

AID Application ID: eindeutige ID des NFC-Readers 16

NFC Near Field Communication: auf RFID basierender Übertragungsstandard 3

RFID radio-frequency identification: Technologie zur kontaktlosen Datenübertragung mittels elektromagnetischer Wellen 5

NDEF NFC Data Exchange Format, Standard-Datenaustauschformat zwischen NFC-Chips . . 5

Refactoring Begriff aus dem Software Engineering, Restrukturierung des Quellcodes zur besseren Lesbarkeit 13

Continuous Integration kontinuierliches Erstellen (Kompilieren) der Anwendung zur Steigerung der Softwarequalität 13

10.2 Quellenverzeichnis

Literatur

[1] Martijn Coenen. *Android-NFC-Forum-Developer-Spotlight-2.pdf*. URL: http://nfc-forum.org/wp-content/uploads/2014/03/Android-NFC-Forum-Developer-Spotlight-2.pdf (besucht am 27. 11. 2015).

[2] Nikolay Elenkov. *Accessing the embedded secure element in Android 4.x*. URL: http://nelenkov.blogspot.nl/2012/08/accessing-embedded-secure-element-in.html (besucht am 27. 11. 2015).

[3] Google. *Host-based Card Emulation | Android Developers*. URL: https://developer.android.com/guide/topics/connectivity/nfc/hce.html (besucht am 27. 11. 2015).

[4] ifross. *GPL Version 2 | ifrOSS*. URL: http://ifross.org/gpl-version-2 (besucht am 19. 01. 2016).

[5] ISO. *ISO 7816-4: Interindustry Command for Interchange ISO7816 4 smart card standard*. URL: http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4.aspx (besucht am 18. 01. 2016).

[6] NXP. "NTAG210/212". In: (). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.434.4624&rep=rep1&type=pdf (besucht am 27. 11. 2015).

[7] Adrian Stabiszewski. *Peer-to-peer communication using NFC in Android 4.4*. URL: http://blog.opendatalab.de/hack/2013/11/25/android-p2p-nfc/ (besucht am 27. 11. 2015).

10.3 Programmcode Dokumentation

Klassen-Dokumentation

10.3.1 addressbookwork.AddressBookReader Klassenreferenz

Öffentliche Methoden

- **AddressBookReader** (MainActivity mainActivity)
- void getAdressbookData (int position)
- void readAllContacts (ContentResolver contentResolver)
- ArrayList< String > **getListItems** ()

10.3.1.1 Ausführliche Beschreibung

Diese Klasse beinhaltet alle Methoden zum Auslesen eines Kontaktes aus dem Adressbuch.

Autor

Klaus Steinhauer, Marko Klepatz

10.3.1.2 Dokumentation der Elementfunktionen

10.3.1.2.1 void addressbookwork.AddressBookReader.getAdressbookData (int *position*)

Diese Methode sammelt beim Aufruf alle benötigten Daten zum ausgewählten Kontakt.

Parameter

position	übergibt die Position aus der Namen-ListView zum lokalisieren des Kontaktes im Adressbuch

10.3.1.2.2 void addressbookwork.AddressBookReader.readAllContacts (ContentResolver *contentResolver*)

Diese Methode liest alle Kontakte im Adressbuch aus und schreibt die Namen in die statische **listItems** Liste aus der **ContactListDialog** Klasse.

Parameter

content↩ Resolver	

Siehe auch

ContactListDialog

10.3.2 addressbookwork.AddressBookWriter Klassenreferenz

Öffentliche, statische Methoden

- static void writeContact (final Context context, final ArrayList< String > cardContent)

10.3.2.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methode zum Schreiben von Address-Daten in das Adressbuch.

Autor

Klaus Steinhauer, Marko Klepatz

10.3.2.2 Dokumentation der Elementfunktionen

10.3.2.2.1 static void addressbookwork.AddressBookWriter.writeContact (final Context *context,* final ArrayList< String > *cardContent*) [static]

Diese Methode generiert einen Intent welcher alle zu schreibenden Address-Daten enthält und ruft im Anschluss das Adressbuch über diesen Intent auf und übergibt die Daten.

Parameter

context	übergibt den Context der rufenden Activity
cardContent	übergibt die zu schreibenden Daten in Form einer Array List

10.3.3 tools.BeamTools Klassenreferenz

Öffentliche Methoden

- **BeamTools** (MainActivity mainActivity)
- void startBeamMode (NfcAdapter nfcAdapter)

10.3.3.1 Ausführliche Beschreibung

Created by ohli on 14.01.16.

10.3.3.2 Dokumentation der Elementfunktionen

10.3.3.2.1 void tools.BeamTools.startBeamMode (NfcAdapter *nfcAdapter*)

Diese Methode leitet den Beamvorgang ein.

Parameter

nfcAdapter	übergibt die NFC-Schnittstelle und versetzt diese in den Beam-Modus.

10.3.4 cardwork.CardReader Klassenreferenz

Öffentliche, statische Methoden

- static String readTag (Intent intent)

10.3.4.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methode zum Auslesen der Kartendaten.

Autor

Oliver Friedrich, Marko Klepatz

10.3.4.2 Dokumentation der Elementfunktionen

10.3.4.2.1 static String cardwork.CardReader.readTag (Intent *intent*) [static]

Diese Methode liest aus dem übergebenem Intent die Daten aus, die sich auf der Karte befinden und wandelt sie in einen String um.

Parameter

intent	enthält die Daten die auf der Karte sind

Rückgabe

gibt den String mit den Daten auf der Karte zurück

10.3.5 cardwork.CardWriter Klassenreferenz

Öffentliche Methoden

- **CardWriter** (Context context)
- void writeNdefMessage (Tag tag, NdefMessage ndefMessage)
- NdefMessage createNdefMessage (String content)

10.3.5.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methoden die für das Beschreiben eines NFC Chips benötigt werden.

Autor

Oliver Friedrich, Marko Klepatz

10.3.5.2 Dokumentation der Elementfunktionen

10.3.5.2.1 NdefMessage cardwork.CardWriter.createNdefMessage (String *content*)

Diese Methode generiert die NDEF Message mit dem speziellen Mime Type für diese App.

Parameter

content	übergibt den Inhalt der auf die Karte geschrieben werden soll

Rückgabe

gibt die generierte NDEF Message zurück

10.3.5.2.2 void cardwork.CardWriter.writeNdefMessage (Tag *tag,* NdefMessage *ndefMessage*)

Diese Methode schreibt die NDEF Nachricht auf den NFC Chip.

Parameter

tag	übergibt das Tag-Format
ndefMessage	übergibt die NDEF Nachricht zum beschreiben auf den NFC Chip

10.3.6 dialogs.ContactListDialog Klassenreferenz

Öffentliche Methoden

- ContactListDialog (MainActivity mainActivity)
- void showDialog ()

10.3.6.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methoden zum Initialisieren und Anzeigen des Kontakt-Listen-↩ Dialogs.

Autor

Marko Klepatz

10.3.6.2 Beschreibung der Konstruktoren und Destruktoren

10.3.6.2.1 dialogs.ContactListDialog.ContactListDialog (MainActivity *mainActivity*)

Dieser Konstruktor leitet alle Initialisierungen für den ContactListDialog ein.

Parameter

mainActivity	übergibt die Klasse MainActivity für den Context der GUI Elemente

Siehe auch

MainActivity
AddressBookReader

10.3.6.3 Dokumentation der Elementfunktionen

10.3.6.3.1 void dialogs.ContactListDialog.showDialog ( )

Diese Methode leitet das Auslesen der Kontakte ein und ruft die Anzeigemethode für den ContactListDialog.

10.3.7 tools.ContactTools Klassenreferenz

Öffentliche Methoden

- ContactTools (MainActivity mainActivity)
- void mailContact (final String email)
- void **easterEgg** (String name)
- void callContact (String name, final String number)

10.3.7.1 Ausführliche Beschreibung

Diese Klasse beinhaltet verschiedene Methoden um die Kontaktdaten direkt an andere Anwendungen im Gerät zu übergeben und zu starten.

Autor

Marko Klepatz

10.3.7.2 Beschreibung der Konstruktoren und Destruktoren

10.3.7.2.1 tools.ContactTools.ContactTools (MainActivity *mainActivity*)

Parameter

mainActivity	übergibt die MainActivity für den Context zum Ausführen von Code der nur auf dieser ausgeführt werden kann

Siehe auch

MainActivity

10.3.7.3 Dokumentation der Elementfunktionen

10.3.7.3.1 void tools.ContactTools.callContact (String *name,* final String *number*)

Diese Methode generiert einen Intent welcher die Nummer des Kontaktes übergibt und ruft mit diesem direkt das Programm zum Anrufen von Kontakten auf und ruft diesen an.

Parameter

name	übergibt den Namen des ausgewählten Kontakts für die Sprachausgabe
number	übergibt die Nummer des ausgewählten Kontakts zum Anrufen

10.3.7.3.2 void tools.ContactTools.mailContact (final String *email*)

Diese Methode generiert einen Intent der die E-Mail Adresse des aktuell ausgewählten Kontakts übergibt und startet eine Auswahl mit allen potenziell einsetzbaren E-Mail Programmen auf dem Gerät.

Parameter

email	übergibt die E-Mail Adresse des aktuell ausgewählten Kontakts

10.3.8 activity.CreateVCardActivity Klassenreferenz

Abgeleitet von AppCompatActivity.

Geschützte Methoden

- void onCreate (Bundle savedInstanceState)
- void onNewIntent (Intent intent)
- void onResume ()
- void onPause ()

10.3.8.1 Ausführliche Beschreibung

Diese Activity beinhaltet die Logik zu den GUI Elementen, die benötigt werden um NFC Medien zu beschreiben.

Zusätzlich beinhaltet sie eine Methode zum Einlesen gültiger NFC Medien.

Autor

Marko Klepatz, Oliver Friedrich

10.3.8.2 Dokumentation der Elementfunktionen

10.3.8.2.1 void activity.CreateVCardActivity.onCreate (Bundle *savedInstanceState*) [protected]

Diese Methode setzt alle relevanten Eigenschaften für die GUI und leitet die Initialisierungen aller Objekte ein.

Zusätzlich nimmt sie den aufrufenden Intent von der MainActivty entgegen und prüft ob dieser Daten zum Beschreiben auf ein NFC Medium enthält und leitet diese weiter, an die GUI Objekte.

Parameter

savedInstanceState	

Siehe auch

NfcAdapter
CardWriter

10.3.8.2.2 void activity.CreateVCardActivity.onNewIntent (Intent *intent*) [protected]

Diese Methode empfängt einen von dem Manifest gefilterten Intent, welcher nur dann in dieser ankommt wenn das NFC Medium, was an das Gerät gehalten wird, mit den gültigen Technologien ausgestattet ist.

Die aktuell gültigen Technologien sind in folgenden Dateien einsehbar:
AndroidManifest.xml
tech.xml

Parameter

intent	

10.3.8.2.3 void activity.CreateVCardActivity.onPause ( ) [protected]

Diese Methode deaktiviert den NFC Adapter.

10.3.8.2.4 void activity.CreateVCardActivity.onResume ( ) [protected]

Diese Methode konfiguriert und aktiviert bei Aufruf den NFC Adapter.

10.3.9 tools.DataWork Klassenreferenz

Öffentliche Methoden

- **DataWork** (Context context)

Öffentliche, statische Methoden

- static String readSingleLineFile (String dataName)
- static List< String > readMultiLineFile (String dataName)
- static void writeSingleLineFile (String dataName, String information)
- static void writeMultiLineFile (String dataName, List< String > informationList)

10.3.9.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methoden zum Lesen und Schreiben von Dateien für die Anwendung.

Autor

Marko Klepatz

10.3.9.2 Dokumentation der Elementfunktionen

10.3.9.2.1 static List<String> tools.DataWork.readMultiLineFile (String *dataName*) [static]

Diese Methode liest eine Datei mit mehreren Zeilen aus und gibt den Inhalt on Form einer Liste vom Typ String zurück.

Parameter

dataName	übergibt den Namen der zu lesenden Datei

Rückgabe

values gibt die Liste mit dem Inhalt jeder Zeile zurück

10.3.9.2.2 static String tools.DataWork.readSingleLineFile (String *dataName*) [static]

Diese Methode liest eine Datei mit einer Zeile aus und gibt den Inhalt on Form eines Strings zurück.

Parameter

dataName	übergibt den Namen der zu lesenden Datei

Rückgabe

gibt den ausgelesenen String zurück

10.3.9.2.3 static void tools.DataWork.writeMultiLineFile (String *dataName,* List< String > *informationList*) [static]

Diese Methode schreibt eine mehrzeilige Information in eine Datei mit Umbrüchen

Parameter

dataName	übergibt den Namen der zu schreibenden Datei
information↩ List	übergibt die zu schreibenden Informationen

10.3.9.2.4 static void tools.DataWork.writeSingleLineFile (String *dataName,* String *information*) [static]

Diese Methode schreibt eine einzeilige Information in eine Datei ohne Umbrüche

Parameter

dataName	übergibt den Namen der zu schreibenden Datei
information	übergibt die zu schreibende Information

10.3.10 activity.MainActivity Klassenreferenz

Abgeleitet von AppCompatActivity, CreateNdefMessageCallback und OnInitListener.

Öffentliche Methoden

- void setVCardInformationOnMainScreen (String vCardInformation)
- NdefMessage createNdefMessage (NfcEvent event)
- void onInit (int status)
- boolean **onCreateOptionsMenu** (Menu menu)
- boolean **onOptionsItemSelected** (MenuItem item)

Statische öffentliche Attribute

- static final String **TAG** = "Nfc Card App"

Geschützte Methoden

- void onCreate (Bundle savedInstanceState)
- void onNewIntent (Intent intent)
- void onResume ()
- void onPause ()
- void onDestroy ()

10.3.10.1 Ausführliche Beschreibung

Diese Activity beinhaltet die Logik zu den GUI Elementen die benötigt werden um NFC Medien zu lesen und anzuzeigen.
Zusätzlich werden hier das Anzeigen von Kontakten aus dem Adressbuch und das Importieren von neuen Kontakten, sowie die Android Beam-Funktion, eingeleitet. Ebenso wird hier auch die CreateVCardActivity gerufen.

Weiterführend wird immer diese Activity gerufen, wenn eine für diese App gültige NFC Technologie oder der spezielle Mime Type dieser App erkannt wird. (Der Mime Type ist in der Klasse **CardWriter** einsehbar)

Autor

Marko Klepatz, Oliver Friedrich

Siehe auch

CardWriter

10.3.10.2 Dokumentation der Elementfunktionen

10.3.10.2.1 NdefMessage activity.MainActivity.createNdefMessage (NfcEvent *event*)

Diese Callback Methode gibt die zu sendende NDEF Message für den Android NFC Beamer zurück.

Parameter

event	

Rückgabe

gibt die zu sendende NDEF Message für den Android NFC Beamer zurück.

10.3.10.2.2 void activity.MainActivity.onCreate (Bundle *savedInstanceState*) [protected]

Diese Methode setzt alle relevanten Eigenschaften für die GUI und leitet die Initialisierungen aller Objekte ein.

Zusätzlich nimmt diese Methode einen von dem Manifest gefilterten Intent entgegen welcher nur dann in dieser ankommt wenn das NFC Medium, was an das Gerät gehalten wird, mit den gültigen Technologien ausgestattet ist bzw. den Mime Type für diese App besitzt.

Die aktuell gültigen Technologien sind in folgenden Dateien einsehbar:
AndroidManifest.xml
tech.xml

Parameter

savedInstanceState	

10.3.10.2.3 void activity.MainActivity.onDestroy ( ) [protected]

Diese Methode beendet beim Aufruf die TextToSpeech Vorgang/Funktion.

10.3.10.2.4 void activity.MainActivity.onInit (int *status*)

Diese Methode weist der TextToSpeech Klasse die Sprache zu mit welcher dann die Sprachausgabe erfolgt. In diesem Fall wird immer die Standardsprache des Geräts verwendet.

Parameter

status	

10.3.10.2.5 void activity.MainActivity.onNewIntent (Intent *intent*) [protected]

Diese Methode empfängt einen von dem Manifest gefilterten Intent, welcher nur dann in dieser ankommt wenn das NFC Medium, was an das Gerät gehalten wird, mit den gültigen Technologien ausgestattet ist.

Die aktuell gültigen Technologien sind in folgenden Dateien einsehbar:
AndroidManifest.xml
tech.xml

Parameter

intent	

10.3.10.2.6 void activity.MainActivity.onPause ( ) [protected]

Diese Methode deaktiviert den NFC Adapter.

10.3.10.2.7 void activity.MainActivity.onResume ( ) [protected]

Diese Methode konfiguriert und aktiviert bei Aufruf den NFC Adapter.

10.3.10.2.8 void activity.MainActivity.setVCardInformationOnMainScreen (String *vCardInformation*)

Diese Methode lädt die Daten auf die **vCardListView**.

Parameter

vCard↩ Information	übergibt die Informationen die sich auf dem NFC Medium befinden.

10.3.11 tools.NfcTools Klassenreferenz

Öffentliche Methoden

- NfcTools (MainActivity mainActivity, NfcAdapter nfcAdapter)
- void checkNFC ()
- void checkNFCSupport ()

10.3.11.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methoden zum Überprüfen ob die NFC Technologie vorhanden und aktiviert ist.

Autor

Marko Klepatz

10.3.11.2 Beschreibung der Konstruktoren und Destruktoren

10.3.11.2.1 tools.NfcTools.NfcTools (MainActivity *mainActivity,* NfcAdapter *nfcAdapter*)

Parameter

mainActivity	übergibt die MainActivity für den Context
nfcAdapter	übergibt den NfcAdapter zur Überprüfung

Siehe auch

MainActivity
NfcAdapter

10.3.11.3 Dokumentation der Elementfunktionen

10.3.11.3.1 void tools.NfcTools.checkNFC ( )

Diese Methode überprüft ob die NFC Technologie aktiviert ist und gibt eine Warnung in Form einer Toast Nachricht aus wenn sie deaktiviert ist.

10.3.11.3.2 void tools.NfcTools.checkNFCSupport ( )

Diese Methode überprüft ob die NFC Technologie auf dem Gerät vorhanden ist und gibt eine Toast Nachricht aus wenn sie es nicht ist und beendet die App.

10.3.12 dialogs.SettingsDialog Klassenreferenz

Öffentliche Methoden

- SettingsDialog (MainActivity mainActivity)
- void **showDialog** ()
- Switch **getVibrationSwitch** ()
- Switch **getVoiceSwitch** ()

10.3.12.1 Ausführliche Beschreibung

Diese Klasse beinhaltet alle Methoden zum Generieren und Anzeigen des Einstellungs-Dialogs. Weiterführend werden hier auch alle Einstellungen direkt in Dateien auf dem Handy geschrieben.

Siehe auch

DataWork

Autor

Marko Klepatz

10.3.12.2 Beschreibung der Konstruktoren und Destruktoren

10.3.12.2.1 dialogs.SettingsDialog.SettingsDialog (MainActivity *mainActivity*)

Dieser Konstruktor leitet alle Initialisierungen für den SettingsDialog ein.

Parameter

mainActivity	übergibt die Klasse MainActivity für den Context der GUI Elemente

Siehe auch

MainActivity

10.3.13 tools.VCardFormatTools Klassenreferenz

Öffentliche, statische Methoden

- static ArrayList< String > extractCardInformation (String[ ] cardContent)
- static String getFormatedVCardString (String userName, String mobileNumber, String homeNumber, String eMail)

10.3.13.1 Ausführliche Beschreibung

Diese Klasse beinhaltet die Methoden, die für die V-Card Formatierung und das Extrahieren der Informationen aus dem V-Card Format benötigt werden.

Autor

Marko Klepatz, Klaus Steinhauer

10.3.13.2 Dokumentation der Elementfunktionen

10.3.13.2.1 static ArrayList<String> tools.VCardFormatTools.extractCardInformation (String[ ] *cardContent*) [static]

Diese Methode extrahiert die Informationen aus einem V-Card String und bringt sie in ein für diese App sinnvolles Ausgabeformat.

Parameter

cardContent	übergibt die rohen Karten-Daten

Rückgabe

vCardInformationList gibt eine Liste mit allen extrahierten Daten zurück

10.3.13.2.2 static String tools.VCardFormatTools.getFormatedVCardString (String *userName,* String *mobileNumber,* String *homeNumber,* String *eMail*) [static]

Diese Methode formatiert die Roh-Adress-Daten in einen V-Card-Format String.

Parameter

userName	übergibt den Namen des gewählten Kontaktes
mobile↩ Number	übergibt die Mobilnummer des gewählten Kontaktes
homeNumber	übergibt die Festnetznummer des gewählten Kontaktes
eMail	übergibt die E-Mail Adresse des gewählten Kontaktes

Rückgabe

gibt den V-Card-Format String zurück

10.3.14 addons.Vibration Klassenreferenz

Öffentliche Methoden

- Vibration (Vibrator vibrator)

Öffentliche, statische Methoden

- static void **setVibration** (boolean vibration)
- static boolean **isVibration** ()
- static void vibrate ()

10.3.14.1 Ausführliche Beschreibung

Diese Klasse ist für die Vibrationsfunktion, insofern aktiviert, für alle Klassen und Activities zuständig.

Sie muss einmal über ihren Konstruktor initialisiert werden und ist dann für alle Klassen, durch ihren statischen Charakter verfügbar.

Vibrator ist die zu initialisierende Klasse mit der die Vibrationsfunktion ausgeführt werden kann.

Autor

Marko Klepatz

10.3.14.2 Beschreibung der Konstruktoren und Destruktoren

10.3.14.2.1 addons.Vibration.Vibration (Vibrator *vibrator*)

Dieser Konstruktor nimmt die in der MainActivity initialisierte Vibrator Klasse entgegen, die hier benötigt wird um die Vibration auszuführen.

Parameter

vibrator	übergibt die benötigte Vibrator Klasse zum Ausführen der Vibration

10.3.14.3 Dokumentation der Elementfunktionen

10.3.14.3.1 static void addons.Vibration.vibrate ( ) [static]

Diese Methode ist beim Aufruf dafür zuständig, dass das Gerät 25 Millisekunden vibriert.

Zusätzlich wird geprüft ob die Variable **vibration** auf true oder false gestellt ist.

true: vibriert
false: vibriert nicht

10.3.15 addons.Voice Klassenreferenz

Öffentliche Methoden

- Voice (TextToSpeech textToSpeech)

Öffentliche, statische Methoden

- static void speakOut (String message)
- static boolean **isSound** ()
- static void **setSound** (boolean sound)

10.3.15.1 Ausführliche Beschreibung

Diese Klasse ist für die Sprachausgabe, insofern aktiviert, für alle Klassen und Activities zuständig.

Sie muss einmal über ihren Konstruktor initialisiert werden und ist dann für alle Klassen, durch ihren statischen Charakter verfügbar.

Vibrator ist die zu initialisierende Klasse mit der die Vibrationsfunktion ausgeführt werden kann.

Autor

Marko Klepatz

10.3.15.2 Beschreibung der Konstruktoren und Destruktoren

10.3.15.2.1 addons.Voice.Voice (TextToSpeech *textToSpeech*)

Dieser Konstruktor nimmt die in der MainActivity initialisierte TextToSpeech Klasse entgegen, die hier benötigt wird um die Sprachausgabe auszuführen.

Parameter

textToSpeech	übergibt die benötigte TextToSpeech Klasse zum Ausführen der Sprachausgabe

10.3.15.3 Dokumentation der Elementfunktionen

10.3.15.3.1 static void addons.Voice.speakOut (String *message*) [static]

Diese Methode ist beim Aufruf für die Sprachausgabe zuständig.

Zusätzlich wird geprüft ob die Variable **sound** auf true oder false gestellt ist.

true: Sprachausgabe wird ausgeführt
false: Sprachausgabe wird nicht ausgeführt

Parameter

message	übergibt die Nachricht, welche über die Sprachausgabe ausgegeben wird