Android API Levels

Mit jeder neuen Android Version werden auch immer neue Features unterstützt. Mit Android 2.0 (Eclair) wurde Multi-Touch eingeführt, seit Android 2.3 (Gingerbread) ist neu die Kommunikation über NFC (Near Field Communication) möglich und Android 3.0 (Honeycomb) kennt neben Activities auch Fragments, um die UI-Möglichkeiten von Tablets besser unterstützen zu können. Der Programmierer muss bei jeder Applikation entscheiden, auf welcher Version er seine Programme entwickelt, und dabei Vorwärts- und Rückwärtskompatibilität beachten. Wir diskutieren dies in diesem Artikel am Beispiel einer Applikation, welche (optional) für den Austausch von Daten auch NFC verwenden soll.

Dominik Gruntz | dominik.gruntz@fhnw.ch

Android Entwickler werden beim Erzeugen eines neuen Projektes in Eclipse gefragt, für welche Version (welches *Build Target*) die Applikation vorbereitet werden soll. Diese Entscheidung legt fest, welche Bibliotheken bereitgestellt werden. Wählt man hier die neueste Android Version, so können alle neuen Features verwendet werden, aber die Applikation läuft nicht mehr auf Geräten, auf welchen eine Vorgängerversion des Systems installiert ist. Wählt man hingegen eine ältere Version, so sind die neuen Features nicht verfügbar.

Tabelle 1 zeigt die Android Plattformen, die bis heute publiziert worden sind, zusammen mit ihrer Verteilung im Markt. Implementiert man auf der Basis von Android 2.1 (Eclair), dann erreicht man 97.2% aller Android Nutzer. Falls man jedoch Android 2.3 (Gingerbread) voraussetzt, dann erreicht man nur noch einen Drittel der Android Nutzer. Eine historische Entwicklung der Verteilung der verschiedenen Android-Versionen findet man unter [And11a].

Vorwärtskompatibilität

Da wir möglichst viele Kunden mit unserer Applikation erreichen wollen, haben wir unser Projekt auf API Level 7 (Android 2.1.x) entwickelt. Google garantiert, dass eine Applikation, die für ein bestimmtes API Level entwickelt worden ist, auch auf allen höheren API Levels ausgeführt werden kann (ohne diese neu zu kompilieren). Dies wird erreicht, in dem bei neuen Android Versionen keine Änderungen vorgenommen werden, welche zu Inkompatibilitäten führen können. Konkret heisst dies, dass nur folgende Änderungen möglich sind:

- Hinzufügen von neuen Paketen, neuen Klassen und neuen Methoden
- Entfernen von Exceptions aus *throws*-Deklarationen in Methoden und Konstruktoren
- Hinzufügen von Runtime-Exceptions in *throws*-Deklarationen
- Verstärkung des Resultattyps bei Methoden (z.B. bei der *clone*-Methode)
- Erweiterung der Sichtbarkeit von Klassen, Methoden, Konstruktoren oder Feldern
- Markierung von Klassen, Konstruktoren, Methoden oder Feldern als *deprecated*

In der Vergangenheit konnten diese Bedingungen grösstenteils eingehalten werden. Mit jeder neuen Version wird auch ein *Android API Differences Report* bereitgestellt der zeigt, welche Änderungen am API vorgenommen worden sind (z.B. in [And11b] für die Änderungen zwischen den Versionen Froyo und Gingerbread).

Vorwärtskompatibilität ist enorm wichtig, denn so wird sichergestellt, dass die installierten Applikationen auch nach einem System-Update noch funktionieren.

Platform Version	API Level	VERSION_CODE	Verteilung
Android 1.0	1	BASE	
Android 1.1	2	BASE_1_1	
Android 1.5	3	CUPCAKE	1.0%
Android 1.6	4	DONUT	1.8%
Android 2.0	5	ECLAIR	
Android 2.0.1	6	ECLAIR_0_1	
Android 2.1.x	7	ECLAIR_MR1	13.3%
Android 2.2.x	8	FROYO	51.2%
Android 2.3, 2.3.1, 2.3.2	9	GINGERBREAD	0.6%
Android 2.3.3, 2.3.4	10	GINGERBREAD_MR1	30.7%
Android 3.0.x	11	HONEYCOMB	0.2%
Android 3.1.x	12	HONEYCOMB_MR1	0.7%
Android 3.2	13	HONEYCOMB_MR2	0.5%

Tabelle 1: Plattform Versionen und deren Verteilung [And11a]

Reflection

Unsere Applikation verwaltet Coupons und Vergünstigungen, die via QR-Codes[1] zwischen Nutzern ausgetauscht werden können. Falls die Applikation auf einem Gerät ausgeführt wird, welches Near Field Communication (NFC) unterstützt, so soll auch der Austausch von Coupons via NFC

1 QR-Codes sind zweidimensionale Codes, QR steht für Quick Response.

```
if(getPackageManager().hasSystemFeature("android.hardware.nfc"))  {
  try {
    Class<?> c = Class.forName("android.nfc.NfcAdapter");
    Method getDefaultAdapter = c.getMethod("getDefaultAdapter",
                                  android.content.Context.class);
    nfcAdapter = getDefaultAdapter.invoke(null, this);

    Class<?> clsNdefMessage = Class.forName("android.nfc.NdefMessage");
    Class<?> clsNdefRecord = Class.forName("android.nfc.NdefRecord");

    Object codeRecord = clsNdefRecord.getConstructor(Short.TYPE,
            byte[].class, byte[].class, byte[].class).newInstance(
      (short)2,    // NdefRecord.TNF_MIME_MEDIA
      "application/vnd.fhnw.coupon".getBytes(Charset.forName("US-ASCII")),
      new byte[0],
      itemData
    );

    Object arr = Array.newInstance(clsNdefRecord, 1);
    Array.set(arr, 0, codeRecord);
    ndefPushMessage = clsNdefMessage.getConstructor(arr.getClass()).newInstance(arr);
  } catch(Exception e) {
    nfcAdapter = null;
  }
}
```

Listing 1: Zugriff auf NFC Funktionalität via Reflection

möglich sein. Die auf diesen Geräten vorhandenen NFC Klassen sind jedoch nicht Teil des API Level 7. Der Zugriff auf diese Klassen muss daher mit Reflection gelöst werden.

Das API Level des Gerätes, auf dem die Applikation ausgeführt wird, kann (seit API Level 4) über den Wert von *Build.VERSION.SDK_INT* ausgelesen werden. Mit dieser Information weiss man, welche zusätzlichen Klassen via Reflection verfügbar sind. Im Fall von NFC kann auch abgefragt werden, ob der entsprechende Service vorhanden ist (denn die NFC Funktionalität ist in Gingerbread optional, d.h. die Klassen sind im API vorhanden, aber es ist nicht zwingend auch ein NFC-Adapter verbaut). In unserem Code prüfen wir daher in der *onCreate* Methode mit dem Aufruf `getPackageManager().hasSystemFeature("android.hardware.nfc")`, ob NFC unterstützt wird.

Listing 1 zeigt den Zugriff auf die NFC-Funktionalität via Reflection. Das Feld *nfcAdapter* ist dabei in der Activity als Feld vom Typ *Object* deklariert und und wird mit dem Default-NFC-Adapter initialisiert. Zusätzlich erzeugen wir noch die NDEF-Meldung, die wir dann via NDEF-Push bereitstellen wollen und legen diese im Feld *ndefPushMessage* vom Typ *Object* ab. In den Methoden *onResume* und *onPause* (in Listing 1 nicht gezeigt) prüfen wir dann, ob dieses Feld definiert ist, bevor wir versuchen, NDEF-Push mit den Methoden *enableForegroundNdefPush* bzw. *disableForegroundNdefPush* zu aktivieren bzw. zu deaktivieren.

Im Manifest haben wir die für NFC nötigen Erweiterungen vorgenommen. Android ignoriert

```
<uses-permission android:name="android.permission.NFC"/>

<application android:icon="@drawable/icon"
             android:label="@string/app_name">
  <activity android:name=".ItemListActivity"
            android:label="@string/app_name"
            android:launchMode="singleTask">

    <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" />
      <data android:mimeType="application/vnd.fhnw.coupon" />
    </intent-filter>
  </activity>
</application>
```

Listing 2: Manifest mit NFC Deklarationen

Permissions und Intent-Filter, die nicht bekannt sind. Das Manifest in Listing 2 funktioniert auf jeden Fall auch unter Android 2.1 (Eclair MR1). Im LogCat erscheint lediglich die Warnung

```
WARN/PackageManager(59):
 Unknown permission android.permission.NFC
  in package ch.fhnw.imvs.android.coupons
```

Rückwärtskompatibilität

Programmierung mit Reflection macht nicht wirklich Spass, daher empfiehlt Google, jeweils gegen das neueste API zu programmieren, dabei jedoch auch ältere APIs zu unterstützen. Dies kann erreicht werden, indem im Manifest mit der *uses-sdk*-Angabe sowohl ein minimales als auch ein Ziel-API spezifiziert wird:

```
<uses-sdk
  android:minSdkVersion="7"
  android:targetSdkVersion="10" />
```

Damit stehen die Bibliotheken des Ziel-APIs (hier Version 10) zur Verfügung, aber die Applikation kann auch bereits ab der Min-SDK-Version (hier Version 7) ausgeführt werden. Auf einem Gerät mit einem älteren Android System kann die Applikation jedoch nicht mehr installiert werden.

Klassen und Methoden, die in einer Version > 7 eingeführt worden sind, dürfen nur verwendet werden, wenn die Applikation auf einem Gerät mit entsprechender Android-Version ausgeführt wird. Wird diese Bedingung verletzt, dann wirft die Dalivk-VM einen *NoClassDefFoundError* und der Benutzer sieht eine Maske wie in Abbildung 1 und ist gezwungen, den Task zu schliessen.

Da jedoch mit den Klassen aus dem Ziel-API gearbeitet werden kann, vereinfacht sich der Code gegenüber jenem aus Listing 1 gewaltig. Im Gegensatz zur Version mit Reflection kann der Compiler nun prüfen, ob die Aufrufe semantisch korrekt sind. Das Resultat findet man in Listing 3. Dieser Code funktioniert auf allen Plattformen, da nur dann auf die NFC Funktionalität zugegriffen wird, falls diese vorhanden ist, und da die Dalvik-VM Klassen erst dann lädt, wenn sie auch benötigt werden. Eine Klasse wird geladen, gelinkt und initialisiert, unmittelbar bevor

- eine Instanz dieser Klasse erzeugt wird,
- eine statische Methode dieser Klasse aufgerufen wird oder
- auf ein statisches Feld dieser Klasse zugegriffen wird (ausser der Zugriff auf Konstanten, d.h. statische Felder vom Typ String oder mit einem primitiven Typ welche final deklariert sind).

Insbesondere dürfen Felder vom Typ *NfcAdapter* oder *NdefMessage* in der Activity deklariert werden, auch wenn diese Klassen erst mit API Level 9 eingeführt worden sind. Die Initialisierung dieser Felder mit *null* oder der Vergleich mit *null* impliziert kein Laden der entsprechenden Klassen und kann damit auch auf einem Gerät mit API Level < 9 ausgeführt werden. Auch die Konstante *PackageManager.FEATURE_NFC* (auch erst seit API Level 9) darf im Code verwendet werden, denn diese Konstante (mit dem Wert "android.hardware.nfc") wird direkt vom Compiler zur Übersetzungszeit aufgelöst.

Der Nachteil dieses Ansatzes ist, dass der Entwickler auch unbeabsichtigt Features aufrufen kann, welche auf der *minSdkVersion* gar noch nicht vorhanden waren. Eclipse zeigt zwar (durch Einblenden der JavaDoc) an, ab welchem API Level eine Methode gültig ist, aber es liegt in der Verantwortung des Programmierers, darauf zu achten. Insbesondere muss die Applikation auf allen unterstützen API-Levels getestet werden. In den Run- bzw. Debug-Konfigurationen von Eclipse kann angegeben werden, welches virtuelle Device beim Start verwendet werden soll. Hier ist die Option «manuell» zu wählen, damit man dann beim Start der Applikation angeben kann, auf welchem API Level die Applikation ausgeführt werden soll.

Um verschiedene Andorid Versionen zu unterstützen wird typischerweise mit dem State-Pattern gearbeitet, d.h. es werden unterschiedliche Klassen implementiert, welche die Funktionalität je nach Build-Version bereitstellen. In einer Factory wird dann beim Programmstart die richtige Strategie erzeugt. Ein Beispiel dazu findet man in [Bray10].

Library-Projekte: leider nein…

Man könnte dieses Problem entschärfen, indem man das Projekt gegenüber einem tieferen Ziel-API entwickelt, und die Zusatzfunktionalität in ein separates Projekt auslagert. Android bietet dazu spezielle Library-Projekte an. Library-Projekte enthalten Code und Ressourcen, die aus mehreren Projekten referenziert werden können. Diese Projekte werden nicht in ein APK-File übersetzt und können entsprechend nicht auf einem Gerät installiert werden, sondern die Klassen dieser Projekte werden in das referenzierende Projekt übernommen. Leider werden sie dabei auch unter dem API Level dieses Projektes übersetzt. Es ist also nicht möglich, ein Library-Projekt mit API Level 10 in ein Projekt mit API Level 7 zu importieren, falls das Library-Projekt Features von

Abbildung 1: NoClassDefFoundError

```
private NfcAdapter nfcAdapter;
private NdefMessage ndefPushMessage;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  // ...
  if(getPackageManager().hasSystemFeature(PackageManager.FEATURE_NFC)) {
    try {
      nfcAdapter = NfcAdapter.getDefaultAdapter(this);
      assert nfcAdapter != null; // guaranteed by spec

      NdefRecord codeRecord = new NdefRecord(
        NdefRecord.TNF_MIME_MEDIA,
        "application/vnd.fhnw.coupon".getBytes(Charset.forName("US-ASCII")),
        new byte[0],
        itemData
      );
      ndefPushMessage = new NdefMessage(new NdefRecord[]{codeRecord});
    } catch(Exception e) {
      e.printStackTrace();
      nfcAdapter = null;
    }
  }
}

@Override
protected void onResume() {
   super.onResume();
   if(nfcAdapter != null) {
     nfcAdapter.enableForegroundNdefPush(this, ndefPushMessage);
   }
}

@Override
protected void onPause() {
  super.onPause();
  if(nfcAdapter != null) {
    nfcAdapter.disableForegroundNdefPush(this);
  }
}
```

Listing 3: Rückwärtskompatibler Zugriff auf NFC

API Level 10 nutzt die im API Level 7 noch nicht vorhanden waren.

Zusammenfassung

Android unterstützt Vorwärtskompatibilität, in dem es bei neuen Android Versionen keine Änderungen vornimmt, welche zu Inkompatibilitäten führen. Um auch ältere Versionen unterstützen zu können, ist im Manifest neben dem Ziel-API-Level auch ein Minimum-API-Level anzugeben. Die Übersetzung erfolgt dann gegenüber dem Ziel-API und es liegt in der Verantwortung des Programmierers sicherzustellen, dass Features nur dann verwendet werden, wenn sie auf dem Gerät, auf dem die Applikation läuft, auch vorhanden sind. Es wäre nützlich, wenn die IDE den Entwickler dabei unterstützen würde, z.B. indem alle Methodenaufrufe und Feldzugriffe, welche erst nach dem Minimum-API-Level eingeführt worden sind, entsprechend markiert würden.

Referenzen

[And11a] http://developer.android.com/resources/dashboard/platform-versions.html (2. September 2011)

[And11b] Android API Differences Report, From Level 8 to Level 9: http://www.devdiv.com/android/docs/sdk/api_diff/9/changes.html

[Bray10] Tim Bray, How to have your (Cup)cake and eat it too: http://android-developers.blogspot.com/2010/07/how-to-have-your-cupcake-and-eat-it-too.html