entwickler.press

Android

Schnelleinstieg

Sven Haiges

schnell + kompakt

Sven Haiges

Android
Schnelleinstieg

schnell+kompakt

entwickler.press

Sven Haiges
Android
Schnelleinstieg
schnell+kompakt
ISBN: 978-3-86802-067-0

ein Imprint der Software & Support Media GmbH

http://www.entwickler-press.de
http://www.software-support.biz

Ihr Kontakt zum Verlag und Lektorat: lektorat@entwickler-press.de

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Lektorat: Sebastian Burkart
Korrektorat: Nataliya Korn
Satz: Pobporn Fischer
Umschlaggestaltung: Maria Rudi
Belichtung, Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, Paderborn.

Inhaltsverzeichnis

Vorwort

Das mobile Betriebssystem Android konnte 2010 einen schier unglaublichen Erfolg verzeichnen. Laut Gartner [1] stieg die Zahl der mit Android verkauften Smartphones um 888,8 %. Damit hat Android Ende 2010 bereits Platz 2 der mobilen Betriebssysteme eingenommen. Apples iOS wurde bereits überholt. Auf Platz 1 befand sich 2010 noch Symbian, jedoch wird erwartet, dass 2011 Android auf Platz 1 der mobilen Betriebssysteme vorrücken wird.

Dies ist vor allem den zahlreichen High-End Produkten von HTC, Samsung und Motorola zu verdanken. Mittlerweile gibt es weltweit kaum einen Mobilfunkanbieter, der es sich leisten kann, kein Android-basiertes Smartphone in seinem Programm zu haben.

Für Entwickler eröffnet sich mit Android eine faszinierende Welt. Dieses Buch möchte Ihnen den Einstieg in die Android-Entwicklung so einfach wie möglich machen.

Eines jedoch gleich vorweg: Android ist ein großes, recht umfassendes Thema und alle paar Monate stellt Google weitere APIs zukünftiger Versionen vor. Für dieses Buch haben wir uns deshalb Bereiche der Android-Entwicklung herausgesucht, die unserer Meinung nach elementar sind. Abgerundet wird dieser Einstieg durch Kapitel zu Maps, Widgets und NFC (Near-Field-Communication). Wir hoffen, dass wir damit den richtigen Mix aus Grundlangen und faszinierenden Zukunftsthemen gefunden haben.

Dieses Buch setzt allerdings auch einige Grundlagen voraus. Beispielsweise beschreiben wir nicht, wie Sie Eclipse und das Android-Development-Tools-(ADT-)Plug-in installieren (siehe [2]). Wir sind der Meinung, dass die meisten diese Grundlagen bereits besitzen.

Feedback und Anregungen sind jederzeit willkommen – per E-Mail (sven.haiges@gmail.com) oder Twitter (@hansamann).

Viel Spaß bei der Lektüre!

Links & Literatur

[1] Gartner Mobile Devices Sales 2010:
http://www.gartner.com/it/page.jsp?id=1543014

[2] Android Eclipse Plugin:
http://developer.android.com/sdk/eclipse-adt.html

Vorbemerkungen

- Zu jedem Kapitel können Sie unter: *www.entwickler-press.de/android* ein ZIP-File mit dem kompletten Quellcode herunterladen.
- Viele Abbildungen und Screenshots im Buch wurden um 90 Grad gedreht, um die Lesbarkeit zu verbessern.

Komponenten einer Android-Applikation

Bevor wir uns dem Hauptthema dieses Kapitels – den Android Activities – widmen, sollten wir wenigstens kurz die unterschiedlichen Komponenten einer Android-Applikation vorstellen und ein paar weitere wichtige Basics erklären. Im Vergleich zu JME- oder iOS-Applikationen bestehen Android-Applikationen nämlich aus meist vielen, lose gekoppelten Teilen. Dies können allen voran die so genannten Aktivitäten (Activities) sein, die dem Benutzer das UI darstellen, oder es können Android Services, Content-Provider, Intents, Broadcast Receiver usw. sein. Da gibt es eine Menge neuer Konzepte zu verstehen, und auch wenn wir gerade am Anfang nicht die Zeit haben, jedes einzelne Konzept umfassend zu behandeln, wollen wir Ihnen einige doch kurz vorstellen:

- **Activities** beherbergen die Präsentationsschicht einer jeden Android-Applikation. Eine Applikation kann dabei beliebig viele Aktivitäten haben und zwischen diesen hin- und herwechseln.

Das UI selbst wird durch Android-UI-Komponenten realisiert, die von der Klasse *android.view.View* erben. Die so genannten ViewGroups sind dafür verantwortlich, die Views (auch Widgets oder Controls genannt) entsprechend zu positionieren.

- **Services** sind die unsichtbaren Helfer Ihrer Applikation, die je nach Applikationslogik regelmäßig erwachen und beispielsweise nach neuen E-Mails schauen. Services selbst haben kein UI, können jedoch durch das Android-Notification-Framework den Nutzer informieren.

- **Intents** (deutsch: Absichten) sind ein enorm wichtiges Konzept. Sie sind nichts anderes als eine Art „Nachricht", die entweder direkt an einen Service oder eine Aktivität adressiert (Explicit Intents) oder systemweit wie eine Art Rundfunk (Broadcast) verteilt werden kann (Implicit Intents). Das Android-Betriebssystem ist in letzterem Fall dafür verantwortlich, potenzielle Empfänger dieser Nachrichten zu finden und dem Anwender eine Auswahl der Applikationen zu präsentieren, die die Nachricht verarbeiten können.

- **Broadcast Receiver** konsumieren die impliziten Nachrichten (Implicit Intents). Hat Ihre Applikation entsprechende Broadcast Receiver registriert, kann ein entsprechender Intent Ihre Applikation starten, um dann die Nachricht zu verarbeiten.

- **Content-Provider**: Mittels eines oder mehrerer Content-Provider kann Ihre Applikation Daten anderen Applikationen zugänglich machen. Auch intern benutzt Android Content-Provider, um beispielsweise die Kontakte des Telefonbuchs anderen Applikationen zur Verfügung zu stellen. Content-Provider benutzen intern meist SQLite-Datenbanken, können jedoch auch andere Mechanismen der Datenhaltung (Dateisystem, Netzwerk) benutzen.

Neben diesen Bestandteilen einer Android-Applikation gibt es noch Weitere: Widgets (nicht zu verwechseln mit den Android-UI-Komponenten, die auch oft Widgets genannt werden) sind beispielsweise kleine Applikationen, die direkt auf dem Home Screen der Geräte ausgeführt werden. Live Folders geben ebenso vom Home Screen aus Zugriff auf die Elemente eines Content-Providers und können dadurch beispielsweise Ihre gespeicherten Kontakte zugänglich machen.

1.1 Anatomie einer Android-Applikation

Wie ist eine typische Android-Applikation strukturiert? Sollten Sie noch nie eine „frische" Android-App mithilfe des Eclipse-ADT-Plug-ins erstellt haben, so sehen Sie anhand **Abbildung 1.1**, wie sie strukturiert ist. Im *src*-Verzeichnis befindet sich freilich der Quellcode Ihrer Applikation. Wenn Sie zur Erzeugung der Applikation das ADT-Plug-in verwendet haben, so mussten Sie (**Abb. 1.2**) ein Java Package sowie den Klassennamen der ersten Aktivität angeben. Das Plug-in erstellt in diesem Fall automatisch ein Package-Verzeichnis im *src*-Verzeichnis und erzeugt eine leere Activity-Klasse, auf die wir später noch eingehen.

Abbildung 1.1: Verzeichnislayout einer Android-Applikation

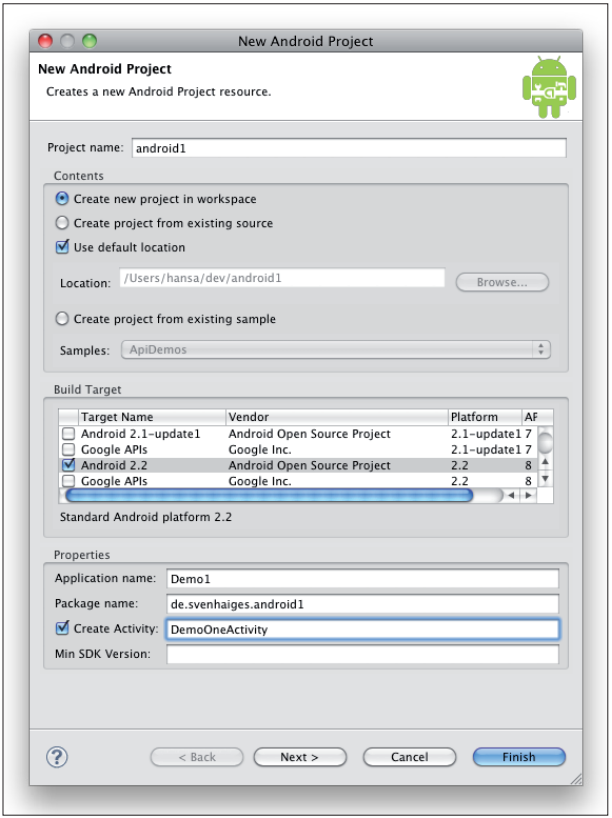

Abbildung 1.2: Erstellen einer Applikation mit dem Eclipse-ADT-Plug-in

Wie der Name des *gen*-Verzeichnisses es schon andeutet, befinden sich in diesem Verzeichnis generierte Dateien. Wenn Sie das Verzeichnis aufklappen, werden Sie eine einzige Klassendatei, *R.java*, vorfinden. Sie enthält Referenzen zu den Ressourcen einer Android-Applikation, die selbst im *res*-Verzeichnis abgelegt werden. Sobald Sie eine neue Ressource, also beispielsweise eine neue Grafikdatei im Verzeichnis *res/drawable* ablegen, sorgt das ADT-Plug-in automatisch dafür, dass in der Klasse *R* eine neue Referenz für diese Ressource angelegt wird. Im Java-Code können Sie das Bild dann per *R.drawable.icon* referenzieren. Ablage und Zugriff von Ressourcen werden auch später in dieser Vorstellung noch detailliert besprochen.

Neben dem *res*-Verzeichnis können im *assets*-Verzeichnis beliebige Dateien abgelegt werden, die somit Bestandteil der Applikation werden. Aus Aktivitäten kann auf dieses Verzeichnis mittels des *AssetManagers* und dem Aufruf *getAssets()* zugegriffen werden. Im Vergleich zum *res*-Verzeichnis können im *assets*-Verzeichnis beliebige Daten abgelegt werden, die keiner von den Android vorgegebenen Strukturen entsprechen müssen.

In Eclipse wird in der Projektansicht nun noch der Eintrag für die Android-Bibliothek *android.jar* angezeigt und *AndroidManifest.xml*. Bei *default.properties* handelt es sich ebenso um eine generierte Datei, die besser nicht angefasst werden sollte. Die Android-Bibliothek *android.jar* befindet sich in Ihrem SDK-Verzeichnis unter *<android_sdk>/platforms/android-8/*. Android-8 steht dabei für das API des Android-Froyo-Releases, hier Android 2.2. Es bleibt noch *AndroidManifest.xml*, die wichtigste Konfigurationsdatei eines jeden Android-Projekts.

1.2 Das Android Manifest – AndroidManifest.xml

Die weiter oben beschriebenen Bestandteile einer Android-Applikation, also Aktivitäten, Services, Content-Provider usw. werden allesamt in der *AndroidManifest.xml* registriert. Glücklicherweise erzeugt das ADT-Plug-in in Eclipse bei der Erstellung einer neuen Android-Applikation diese Datei. Sobald Sie weitere Komponenten hinzufügen, müssen Sie die Konfiguration jedoch selbst erweitern, da Ihre Applikation sonst diese neuen Komponenten nicht finden kann (Listing 1.1).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
  "http://schemas.android.com/apk/res/android"
  package="de.svenhaiges.android1"
  android:versionCode="1"
  android:versionName="1.0">
    <application android:icon="@drawable/icon"
      android:label="@string/app_name">
       <activity android:name=
          "DemoOneActivity"android:
              label="@string/app_name">
        <intent-filter>
         <action android:name=
          "android.intent.action.MAIN" />
         <category android:name=
         "android.intent.category.LAUNCHER" />
        </intent-filter>
       </activity>
    </application>
</manifest>
```

Listing 1.1: „AndroidManifest.xml“ hält die Android-Applikation zusammen

Wenn Sie sich gerade erst von XML verabschiedet haben und Ihre Konfiguration ansonsten nun deklarativ durch Annotations oder wie im Fall diverser Webframeworks (u. a. Grails) im Groovy-Code ablegen, dann müssen Sie bei der Android-Entwicklung wieder etwas umdenken. Android benutzt zumindest während der Entwicklungsphase viel XML, um die Konfiguration und Ressourcen einer Applikation zu beschreiben.

Der Root-Tag *manifest* gibt mit dem Attribut *package* den Package-Namen der Quelldateien an. Somit muss dieser nicht wiederholt werden, was etwas Tipparbeit spart. Betrachten Sie kurz den Tag *activity*, durch den unsere derzeit einzige Aktivität deklariert wird. Das im Namespace *android* befindliche Attribut *name* (also komplett *android:name*) gibt den Klassennamen unserer Aktivität an (und ja, der Punkt vor dem Klassennamen darf nicht weggelassen werden), so wie dies im Eclipse-ADT-Plug-in eingetippt wurde: *DemoOneActivity*.

Der Tag <*intent-filter*> stellt sicher, dass unsere Aktivität zum einen als Haupteinstiegspunkt der Applikation (*android.intent.action.Main)* deklariert wird. Zum anderen wird durch die Kategorie *android.intent.category.Launcher* angegeben, dass diese Aktivität nach der Installation im App-Verzeichnis des Geräts aufgeführt werden soll.

Sind Ihnen die mit @ beginnenden Werte der Attribute *android:icon* und *android:label* auch aufgefallen? Es sind Zeiger auf Android-Ressourcen, also beispielsweise Bilder oder Texte welche im *res*-Verzeichnis abgelegt worden sind. Die Verwendung dieser Android-Ressourcen bringt den Vorteil, dass Android je nach Konfiguration die passende Ressource selbst auswählen kann. Beispielsweise kann so für das Icon, das mit *@drawable/icon* referenziert wird, je nach Auflösung und Größe des Dis-

plays ein passendes Bild ausgewählt werden. Die Konfiguration kann sich auch während der Ausführung einer Applikation ändern, etwa wenn das mobile Gerät ins Querformat gedreht wird. Hierbei möchte man als Entwickler eventuell ein anderes Layout, andere Texte oder Bilder verwenden. Genau dies wird möglich, wenn man konsequent die Android-Ressourcen verwendet.

1.3 Android-Ressourcen

Was genau sind diese Android-Ressourcen? Zunächst sollten wir unsere Aufmerksamkeit ganz dem *res*-Verzeichnis widmen. In diesem werden alle Ressourcen abgelegt. Für jeden Typ wird dafür ein neues Verzeichnis verwendet. Einen Typ, nämlich Grafiken, konnten Sie schon anhand der *AndroidManifest.xml*-Datei sehen. Tabelle 1.1 gibt Ihnen einen kleinen Überblick, was Android-Ressourcen alles sein können.

Typ	Unterverzeichnis	Beschreibung	Zugriff via R.
Einfache Werte	res/values	Strings, einzelne Farbwerte, boolesche Werte, Dimensionen, String/Integer Arrays	R.string, R.color, R.bool, R.dimen, R.array
Grafische Elemente	res/drawable	Bitmaps, NinePatches (stretchable PNGs), Formen sowie Elemente, die aus anderen Elementen zusammengesetzt wurden	R.drawable
Styles und Themes	res/values		R.style

Typ	Unterverzeichnis	Beschreibung	Zugriff via R.
UI-Layouts	res/layout	Layouts für das UI der Applikation	R.layout
Animationen	res/anim und res/drawable	Tween- und Frameanimationen	R.anim und R.drawable
Menüs	res/menu	Inhalte der Applikationsmenüs	R.menu

Tabelle 1.1: Einige der verschiedenen Android-Ressourcen

Tabelle 1.1 sollte Ihnen einen guten Überblick über die wichtigsten Ressourcen geben, die im *res*-Verzeichnis definiert werden können. Für diese und noch weitere Ressourcen können XML-Dateien angelegt werden, welche die Ressource beschreiben. Im einfachsten Fall, für einfache Werte, sieht dies dann folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/values/simple.xml -->
<resources>
    <string name="hello">Hello World,
     DemoOneActivity!</string>
    <string name="app_name">Demo1</string>
    <string-array name="first_names">
        <item>Susanne</item>
        <item>Johannes</item>
    </string-array>
    <bool name="reconnect_often">
      true
    </bool>
    <dimen name="font_size">16sp</dimen>
</resources>
```

Da es sich in diesem Fall allesamt um einfache Werte handelt, können diverse Typen miteinander kombiniert werden. Auch der Dateiname der XML-Datei ist nicht relevant, Hauptsache die Datei befindet sich im *res/values*-Verzeichnis.

Im Codebeispiel definieren wir zunächst zwei einfache Strings, die später im Java-Code mittels *R.string.hello* und *R.string.app_name* referenziert werden können. Der folgende String Array wird per *R.array.first_names*, der boolsche Wert per *R.bool.reconnect_often* und zu guter Letzt die Dimension per *R.dimen.font_size* referenziert. Beachten Sie, dass die automatisch generierte Klasse *R* nicht direkt Objekte vom Typ *String* oder *Integer* enthält, vielmehr enthält diese generierte Klasse lediglich statische Variablen, die allesamt vom Typ *Integer* sind und intern zur Referenzierung der Werte benutzt werden. Um tatsächlich einen String wie *R.string.hello* auszulesen, kann folgender Code benutzt werden: *String hello = getString(R.string.hello);*.

Sobald Sie eine Ressource in einer anderen XML-Datei (sprich Ressource) referenzieren wollen, beispielsweise um die Schriftgröße einer TextView-UI-Komponente zu setzen, muss der entsprechende XML-Syntax benutzt werden, in diesem Fall folgendermaßen:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
```

Mehr zu den unterschiedlichen Ressourcen gibt es unter [3]. Selbstverständlich gehen wir in den nächsten Kapiteln auch auf die passenden Ressourcen ein und stellen diese vor.

1.4 Wieso der ganze Aufwand?

Der wahre Grund, diesen ganzen Aufwand zu betreiben, besteht darin, dass Android so die jeweilige Ressource dynamisch auswählen kann. Ein kleines Beispiel veranschaulicht dies sicherlich am besten. Neben dem Verzeichnis *res/values-* können weitere Verzeichnisse mit *values-* beginnend angelegt werden, und Android sucht automatisch das richtige, passende Verzeichnis aus. Beispielsweise kann dadurch eine Applikation sehr einfach lokalisiert werden:

```
res/
  values/
    strings.xml
  values-de/
    strings.xml
  values-fr
    strings.xml
```

Durch das Anhängen bestimmter Qualifier kann Android je nach Spracheinstellung des Nutzers die passende XML-Datei mit den String-Ressourcen laden. Falls keine spezifische Version passt, so werden per Fallback die Standardressourcen geladen. Diese Auswahl der Ressourcen ist natürlich nicht nur auf Strings oder einfache Werte beschränkt, sämtliche Android-Ressourcen können über diesen Mechanismus weiter qualifiziert werden. Und auch die Qualifier sind nicht nur auf Sprache und Region beschränkt, sondern recht umfassend. Die vollständige Liste aller Qualifier können Sie unter [4] nachlesen, eine Auswahl der wohl wichtigsten Qualifier können Sie in Tabelle 1.2 sehen.

Qualifier	Werte	Beschreibung
MCC und MNC	mcc262, mcc262-mnc01	Der Mobile Country Code (MCC), optional gefolgt vom Mobile Network Code (MNC). Dadurch kann beispielsweise eine Applikation je nach eingebuchtem Netz ein anderes Branding/UI bekommen. Eine Übersicht der MCC und MNC finden Sie unter *http://en.wikipedia.org/wiki/Mobile_Network_Code#G.*
Sprache und Region	de, fr, de-rDE	Die Sprache wird durch die ISO639-1-Codes festgelegt, optional kann die Region (Kürzel siehe ISO 3166-1 alpha 2) durch ein nach dem Bindestrich vorangestelltes kleines „r“ angegeben werden.
Display-größe	small, normal, large	Die physikalische Größe des Displays, die zur Vereinfachung durch die 3 Kategorien small, normal und large ausgedrückt wird.
Display-ausrichtung	port, land	Portait oder Landscape entsprechen der vertikalen oder horizontalen Ausrichtung des Geräts.
Bildschirm Pixeldichte (Pixel Density)	ldpi, mdpi, hdpi, nodpi	Basierend auf der Dichte des Displays (wie viele Punkte pro Inch – dpi) werden drei Gruppen eingeteilt: ldpi mit ca. 120dpi, mdpi mit ca. 160dpi und hdpi mit ca. 240dpi. Nodpi kann benutzt werden, wenn Bitmaps nicht skaliert werden sollen.
System-version	v3, v4, v7, v8 usw.	Die Ressourcen können auch je nach API-Level unterschieden werden. V8 steht dabei für das Android 2.2 Release, also Froyo.

Tabelle 1.2: Qualifier des Android-Resource-Management-Systems