



jetzt lerne ich

Android 4

Programmierung



DIRK LOUIS / PETER MÜLLER

- Grafische Benutzeroberflächen erstellen
- Mehr als 50 fertige Beispiel-Apps entwickeln
- Mit der Eclipse-Entwicklungsumgebung arbeiten
- Alle Tools, Installationsanleitung und 300-seitiges Java-Tutorial auf DVD



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

Die Informationen in diesem Buch werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.
Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt. Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ®-Symbol in diesem Buch nicht verwendet.

10 9 8 7 6 5 4 3 2 1

15 14 13

ISBN 978-3-8272-4818-3 Print; 978-3-86325-562-6 PDF; 978-3-86325-152-9 ePUB

© 2013 by Markt+Technik Verlag,
ein Imprint der Pearson Deutschland GmbH, Martin-Kollar-Straße 10-12, D-81829 München/Germany
Alle Rechte vorbehalten

Covergestaltung: Thomas Arlt, tarlt@adesso21.net

Lektorat: Thomas Pohlmann

Fachlektorat: Frank Langenau

Korrektorat: Petra Alm

Herstellung: Martha Kürzl-Harrison, mkuerzl@pearson.de

Satz: text&form GbR, Fürstenfeldbruck

Druck und Verarbeitung: Drukarnia Dimograf, Bielsko-Biala

Printed in Poland

Dirk Louis, Peter Müller



Jetzt lerne ich

Android 4-Programmierung

Der schnelle Einstieg in die App-Entwicklung
für SmartPhone und Tablet

3 Was wann wofür

- SIE LERNEN IN DIESEM KAPITEL,
- WIE APPS AUFGEBAUT SIND,
 - WELCHE BIBLIOTHEKS-KLASSEN FÜR DIE APP-PROGRAMMIERUNG WICHTIG SIND UND
 - ERFAHREN, WOFÜR MANIFEST-, JAR-, APK- UND ANDERE DATEIEN BENÖTIGT WERDEN.

Nachdem die ersten Schritte getan sind, die erste App erstellt und hoffentlich auch erfolgreich im Emulator getestet wurde, wir also die ersten Hürden erfolgreich genommen haben – und dies gilt ganz besonders für diejenigen Leser, die nebenbei auch noch das Java-Tutorium auf der Buch-DVD durchgearbeitet haben –, werden wir in diesem Kapitel eine kurze Zwischenpause einlegen, die Entwicklerwerkzeuge und den Compiler für eine Weile ruhen lassen und diese Unterbrechung dazu nutzen, uns geistig auf die nächsten Aufgaben vorzubereiten.

Konkret werden wir uns eine Übersicht darüber verschaffen, was bei der App-Programmierung eigentlich von uns erwartet wird, mit welchen Komponenten (Android-Klassen) wir es dabei zu tun haben und wie Android-Projekte im Detail aufgebaut sind. Wir werden dabei etlichem Bekannten begegnen, aber auch viel Neues entdecken.

3.1 Was ist zu tun? – Die drei Pfeiler der App-Erstellung

Zur App-Programmierung gehört, dass Sie

- den **Code schreiben**, der festlegt, wie sich die App verhält,
- das **Layout festlegen**, das bestimmt, wie die App auf dem Anzeigegerät dargestellt wird,
- die **Ressourcen bereitstellen**, die für die Anzeige und Funktion der App benötigt werden.

Haben Sie die Aufgabenkomplexe wiedererkannt? Es sind die gleichen Aufgaben, die wir bereits in Kapitel 2.3 und 2.4 angesprochen haben.

Der Code ist naturgemäß das eigentliche, ureigene Metier des Programmierers. Wie das Codegerüst einer App aussieht, haben Sie ja bereits in Kapitel 2.3 gesehen. In Kapitel 4 werden wir uns etwas näher mit dem Code befassen, ein paar Fingerübungen machen und uns vor allem ansehen, wie uns Eclipse bei der Codebearbeitung unterstützt.

App-Benutzeroberflächen werden üblicherweise über XML-Layoutdateien definiert (siehe Kapitel 2.4). Da sie für den Erfolg und die Bedienbarkeit einer App von großer Bedeutung sind, werden wir uns in Kapitel 5 etwas ausführlicher mit ihnen beschäftigen. Wir werden uns in den XML-Code einarbeiten und uns ansehen, wie uns der Eclipse-Designer bei der visuellen Bearbeitung der Layouts unterstützt, und uns nebenbei mit Hintergrundbildern, Orientierungen und App-Symbolen befassen.

Apps arbeiten viel mit Ressourcen: Bilder, anzuzeigende Texte und Beschriftungen, Farben, Stile, Mediendateien etc. In Kapitel 6 werden wir die verschiedenen Ressourcentypen vorstellen. Vor allem die Arbeit mit Strings und Bildern, inklusive der Unterstützung unterschiedlicher Geräteauflösungen, werden wir etwas eingehender betrachten.

3.2 Wer hilft uns? – Bausteine und Klassen

Android-Apps werden in der Umgebung eines Android-Betriebssystems ausgeführt. Das Betriebssystem stellt, unterstützt von passender Hardware, den Apps viele interessante Optionen zur Verfügung (wie z.B. das Abspielen von Sounddateien, das Aufnehmen von Fotos, das Starten fremder App-Komponenten, das Speichern von Dateien etc.), stellt umgekehrt aber auch Anforderungen an die Apps (vorgegebener Aufbau, vorgegebene Kommunikationswege, DEX-Bytecode etc.).

Um dem Programmierer die Arbeit zu erleichtern, sodass er mit möglichst geringem Aufwand Android-konforme Apps erstellen und die vielen technischen Möglichkeiten nutzen kann, stellt uns Google die Klassen der Android-Bibliothek zur Verfügung.

3.2.1 Bausteine für den App-Aufbau

Apps bestehen aus diversen Bausteinen, hinter denen naturgemäß Klassen aus der Android-Bibliothek stehen. Sehen wir uns einige dieser Bausteine etwas genauer an.

Activities

Während Windows-Anwendungen üblicherweise **ein** Hauptfenster besitzen, in dem der Anwender **eine Vielzahl** von Aktionen durchführen kann, bestehen Apps aus **einer oder mehreren** Bildschirmseiten, die jede **einer** bestimmten Aktivität gewidmet sind. Womit wir beim Thema Aktivitäten, oder, wie der Android-Programmierer auch sagt, »Activities« wären.

Eine Activity ist eine Kombination aus Bildschirmseite und zugehörigem Code. Eine Activity sollte einer in sich abgeschlossenen Aufgabe (Aktivität) gewidmet sein, sie wird als Klasse implementiert, die von der Bibliotheksklasse `android.app.Activity` abzuleiten ist, und sie muss in der Manifestdatei der App aufgeführt werden.

Eine Besonderheit der App-Activities ist, dass sie in sich geschlossene Bausteine darstellen und nur lose an ihre App gebunden sind. Dies hat zwei Konsequenzen:

- Eine Activity kann grundsätzlich von jeder App auf dem Android-Gerät aufgerufen werden.
- Für den Aufruf von Activities gibt es einen globalen Aufrufmechanismus: der Aufruf über Intents. Diesen Mechanismus müssen Sie verwenden, gleichgültig, ob Sie eine Activity der eigenen oder einer fremden App aufrufen möchten.

Intents

Betrachtet man Apps als lose verbundene Activities, so sind es die Intents, zu Deutsch »Absichten«, die die lose Verbindung zwischen den Activities herstellen.

Wiederverwendbare Komponenten

Ist es nicht seltsam, eine Anwendung als eine Sammlung eigenständiger binärer Software-Komponenten zu definieren? Ganz und gar nicht! Microsoft betreibt seit Jahren einen enormen Aufwand, um über diverse Technologien (COM, DCOM, COM+, .NET Framework) seiner Windows-Entwicklergemeinde das zu bieten, was Android von vornherein mitbringt: die einfache anwendungsübergreifende Wiederverwendung von auf dem System installierten Software-Bausteinen.

Konkret bedeutet dies: Wenn Sie aus einer Activity heraus eine andere Activity starten möchten, müssen Sie

- einen Intent erzeugen, der angibt, welche Activity aufzurufen ist und welche Daten dieser Activity gegebenenfalls übergeben werden sollen,
- den Intent mit einer passenden Android-Methode abschicken.

Das Android-System empfängt den Intent und sucht nach der auszuführenden Activity. Gibt es auf dem System eine Activity, die zur Beschreibung in dem Intent passt, wird die Activity gestartet.

Tauchen wir noch ein wenig tiefer in den Intent-Mechanismus ein. Grundsätzlich gibt es zwei Wege, einen Intent zu adressieren:

- als expliziten Intent – in diesem Fall wird als Adressat die Activity angegeben, die aufgerufen werden soll.
- als impliziten Intent – in diesem Fall wird als Adressat keine konkrete Activity angegeben. Stattdessen werden bestimmte Informationen über die gewünschte Aktion mitgeliefert (*action*, *type* und *category*), und das Android-System bestimmt, welche der auf dem System vorhandenen Activities zu den Informationen passt. (Zur Unterstützung dieses Mechanismus definieren die Activities in der Manifestdatei sogenannte Intent-Filter, deren Daten mit den Informationen im Intent-Objekt abgeglichen werden, siehe auch weiter unten die Ausführungen zur Manifestdatei.)

Aufsteiger Activities sind nicht die einzigen Komponenten, die über Intents aufgerufen werden können. Auch Services und Broadcast Receiver sind ausführbare binäre Komponenten, die über Intents gestartet werden.

Broadcast Intents

Nicht nur Apps können Intents abschicken. Auch das Android-System selbst kann Intents versenden, die von interessierten Apps über Broadcast Receiver abgefangen werden können. Wir sprechen in diesem Fall von Broadcast Intents.

Views

Kommen wir noch einmal auf die grafischen Benutzeroberflächen der Apps zurück. Diese sind, wie Sie wissen, auf Bildschirmseiten verteilt und werden üblicherweise über XML-Layoutdateien definiert (siehe auch Kapitel 2.4).

Aufgebaut werden diese Bildschirmseiten aus Views. Eine View, zu Deutsch »Ansicht«, ist einfach ein rechteckiges Element einer UI-Oberfläche, das sich selbst zeichnet und grundsätzlich mit dem Anwender interagieren kann.

Drei Arten von Views sind für uns besonders interessant:

- Zeichenflächen – Instanzen der Klassen `View`, `ImageView` oder `SurfaceView`, die rechteckige Bereiche auf einer Bildschirmseite repräsentieren, in die wir zeichnen können.
- Widgets – spezialisierte Views, die einer bestimmten Aufgabe gewidmet sind. So sind z.B. die typischen Steuerelemente wie Eingabefelder, Schaltflächen, Listenfelder etc. als Widgets im Paket `android.widget` definiert.
- Viewgroups – Container-Views, die andere Views in sich aufnehmen können. Viewgroups, die andere Views nicht nur aufnehmen, sondern auch noch nach bestimmten Regeln anordnen, bezeichnen wir als Layout-Views (siehe Kapitel 5.3).

Sonstige Bausteine

Es gibt noch eine Reihe weiterer App-Bausteine, die in diesem Buch zwar keine besondere Rolle spielen, von denen Sie aber dennoch schon einmal gehört haben sollten.

- Services

Ein Service, zu Deutsch »Dienst«, ist eine Arbeit, die im Hintergrund erledigt wird. Im Gegensatz zu Activities besitzen Services daher keine eigene Benutzeroberfläche. Mithilfe von Services kann man Aufgaben parallel zur laufenden App ausführen (beispielsweise eine Hintergrundmusik abspielen) oder langwierige Aktionen, wie z.B. das Herunterladen großer Multimediadateien aus dem Internet, im Hintergrund erledigen, ohne dass die App dadurch lahmgelegt wird. Services werden als Unterklassen der Klasse `Service` implementiert.

- Broadcast Receiver

Broadcast Receiver sind Komponenten, die auf Meldungen des Android-Systems reagieren, wie z.B. »niedriger Batteriestand« (`ACTION_BATTERY_LOW`) oder »Der Kameraknopf wurde gedrückt« (`ACTION_CAMERA_BUTTON`). Broadcast Receiver besitzen keine eigene Benutzeroberfläche, können aber Nachrichten und Optionen zur Reaktion auf das Ereignis in die System-Statusleiste ausgeben.

- Content Provider

Content Provider sind Komponenten, die Ihnen bei der Verwaltung externer Daten helfen. Sie können sie für Daten benutzen, die nur von einer App verwendet werden. Sie können über einen Content Provider aber auch Daten verwalten, die von mehreren oder allen Apps auf einem Android-Gerät genutzt werden können.

Das Activity-View-Intent-Geflecht

Eine Android-App besteht aus einer oder mehreren Activities. Jede dieser Activities steht für einen in sich abgeschlossenen Aufgabenbereich inklusive zugehöriger Bildschirmseite. Apps können aus prinzipiell beliebig vielen Activities (Bildschirmseiten) bestehen. Der Wechsel von einer Activity zur anderen erfolgt stets durch Absendung eines Intents.

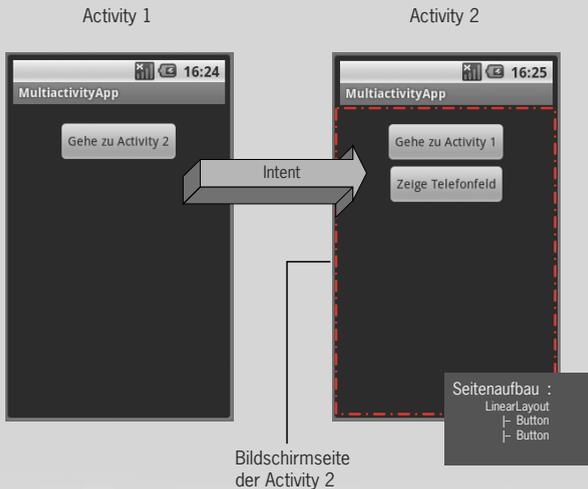


Abbildung 3.1: Drückt der Anwender den Button auf der 1. Bildschirmseite, wird ein Intent abgesetzt, der die Bildschirmseite der 2. Activity aufruft. Will der Anwender zurück zur 1. Seite, muss er auf den zugehörigen Button drücken, der natürlich ebenfalls einen Intent absetzt (hier nicht dargestellt). Die gestrichelte Linie markiert den Bereich, den die Bildschirmseite umfasst.

- Fragmente

Mit Android 3.0 (API-Level 11) eingeführte Komponente. Fragmente erlauben dem Programmierer, den Code umfangreicher Activities aufzuteilen, indem er Teile der Activity in Fragmenten auslagert. Fragmente definieren ihre eigene Benutzeroberfläche und haben einen eigenen Lebenszyklus. Ihr wohl größter Vorzug: auf Fragmente basierende grafische Benutzeroberflächen können so aufgebaut werden, dass funktionell zusammen gehörende Fragmente auf einem Tablet-PC nebeneinander, auf einem Smartphone aber nacheinander angezeigt werden. Eine so konzipierte App bietet auf Tablet-PCs einen größeren Bedienungskomfort, ist aber auch auf Smartphones ausführbar.

App-Komponenten

Activities, Services, Content Provider, Broadcast Receiver und Fragmente werden in der Android-Terminologie als »Android-Komponenten« bezeichnet – essentielle Bausteine, die Aufgaben definieren, die vom Anwender oder vom System gestartet werden können.

Threads und asynchrone Tasks

Wenn Sie im Code einer Activity zeitraubende Aktionen ausführen, wie z.B. das Herunterladen größerer Dateien aus dem Internet, langweilen Sie den Anwender (der warten muss und nicht mit der App fortfahren kann) und alarmieren das Android-System, das irgendwann erkennt, dass die App nicht mehr reagiert (sie ist ja beschäftigt), und dem Anwender eine »Application Not Responding«-Meldung anzeigt.

Sie können dies verhindern, indem Sie die zeitraubende Aktion im Hintergrund ausführen lassen – wahlweise als eigenen Service oder als Thread. Threads können Sie mit der Java-Klasse `java.lang.Thread` oder mithilfe der Android-Klasse `android.os.AsyncTask` implementieren (siehe z.B. Kapitel 18).

DVD zum Buch Für Leser, die mit der Thread-Programmierung unter Java nicht vertraut sind, gibt es auf der Buch-DVD einen Exkurs.

3.2.2 Klassen zur Adressierung spezieller Aufgaben

Neben den grundlegenden App-Bausteinen gibt es in der Android-Bibliothek natürlich noch einen reichen Fundus weiterer Klassen, die Ihre App nutzen kann, um bestimmte Aufgaben zu erledigen, wie z.B.:

Klasse	Für
BitmapFactory Bitmap	Zum Laden, Erzeugen und Arbeiten mit Bildern Paket <code>android.graphics</code>
Camera	Zur Durchführung von 3D-Transformationen auf Grafiken Paket <code>android.graphics</code>
Camera	Zum Zugriff auf die Kamera des Android-Geräts Paket <code>android.hardware</code>
LocationManager Location	Zum Abfragen und Arbeiten mit geografischen Positionen Paket <code>android.location</code>
Log	Zum Ausgeben von Protokollmeldungen Paket <code>android.util</code>
MediaPlayer	Zum Abspielen von Mediadateien Paket <code>android.media</code>
SensorManager	Zum Zugriff auf die Sensoren Paket <code>android.hardware</code>
SoundPool	Zum Abspielen von Signaltönen Paket <code>android.media</code>

Table 3.1: Nützliche Android-Klassen

3.3 Wo wird was gespeichert? – Dateitypen, die Sie kennen sollten

In Kapitel 2 haben wir uns bereits einen groben Überblick über den Aufbau von Android-Projekten und die darin verwalteten Dateien verschafft. Jetzt, da wir bereits ein bisschen Praxis in der App-Programmierung gesammelt haben, wollen wir an dem Gelernten anknüpfen und uns die beteiligten Dateitypen etwas näher ansehen.

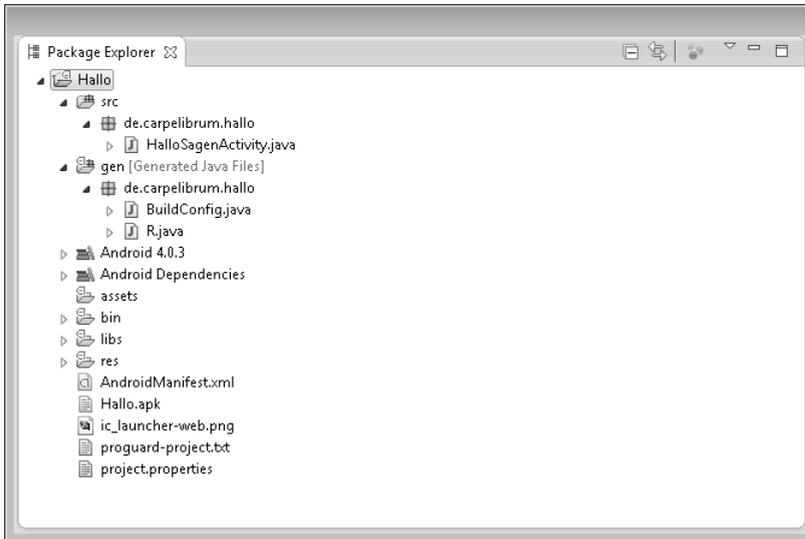


Abbildung 3.2: Aufbau eines Android-Projekts (dargestellt im Package Explorer von Eclipse)

3.3.1 Quelldateien

Android-Apps werden in Java geschrieben, weswegen der Code einer App in JAVA-Dateien im Ordner `src` zu finden ist. Die erste JAVA-Datei für die Start-Activity der App wird automatisch vom Android-Plugin angelegt. Weitere Quelldateien für zusätzliche Activities oder andere Java-Klassen müssen Sie selbst anlegen – vorzugsweise über den Eclipse-Befehl `NEW/CLASS` (siehe auch Kapitel 4.2).

Die Quelldateien der App müssen – wir haben es bereits mehrfach angesprochen – in einem eigenen Paket (hier `de.carpe librum.hallo`) definiert werden. Sie müssen allerdings nicht zwangsweise in einem Paket liegen. Nicht unüblich ist z.B. die Aufteilung der Dateien (und damit der in ihnen definierten Klassen) nach Aufgabenbereichen auf diverse Unterpakete.

Per Doppelklick auf einen Quelldateiknoten im Package Explorer können Sie die Quelldatei zur Bearbeitung in den Eclipse-Editor laden.

Achtung Zur Erinnerung für Java-Neulinge: In einer Java-Quelldatei kann immer nur eine `public` Klasse definiert werden und die Datei muss dann den gleichen Namen tragen wie diese Klasse.

3.3.2 Automatisch generierte Dateien

Im Ordner `gen` steht – neben der für uns nicht weiter interessanten `BuildConfig.java`-Datei – die Datei `R.java`, welche die ID-Konstanten für den Zugriff auf die Ressourcen der App definiert. Diese Datei wird automatisch von Eclipse erstellt und verwaltet. Sie können sie einsehen, indem Sie sie in den Editor laden, aber Sie sollten sie nicht bearbeiten.

Wenn Sie der Ansicht sind, dass die Datei nicht auf dem aktuellen Stand ist, speichern Sie einfach (Befehl `FILE/SAVE ALL`) und erstellen Sie das Projekt danach neu.

Hinweis `R.java` wird im Zuge der Projekterstellung angelegt bzw. aktualisiert.

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package de.carpelibrum.hallo;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int padding_large=0x7f040002;
        public static final int padding_medium=0x7f040001;
        public static final int padding_small=0x7f040000;
    }
    public static final class drawable {
        public static final int ic_action_search=0x7f020000;
        public static final int ic_launcher=0x7f020001;
    }
    public static final class id {
        public static final int menu_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_hallosagen=0x7f030000;
    }
}
```

```
public static final class menu {
    public static final int activity_hallosagen=0x7f070000;
}
public static final class string {
    public static final int app_name=0x7f050000;
    public static final int hello_world=0x7f050001;
    public static final int menu_settings=0x7f050002;
    public static final int title_activity_hallo_sagen=0x7f050003;
}
public static final class style {
    public static final int AppTheme=0x7f060000;
}
}
```

Listing 3.1: Die *R.java*-Datei des App-Projekts *Hallo*

Wir erkennen z.B. die ID `R.layout.activity_hallosagen`, die wir in unserer Hallo-App zum Laden des Layouts aus `activity_hallosagen.xml` benutzt haben:

```
setContentView(R.layout.activity_hallosagen);
```

und die ID des Strings `R.string.hello_world`, der in der `activity_hallosagen.xml`-Layoutdatei als Titel in das `TextView`-Element geladen wird:

```
<RelativeLayout ...>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
    android:text="@string/hello_world"
  />
</RelativeLayout>
```

Mehr zu der *R.java*-Datei und den darin definierten ID-Konstanten finden Sie in Kapitel 6, »Ressourcen«.

Hinweis Apropos automatisch generierte Dateien. Wo stehen eigentlich die CLASS-Dateien, die der Compiler beim Erstellen der App erzeugt? Sie stehen im Unterverzeichnis *bin*, das Sie im Projektverzeichnis auf Ihrer Festplatte finden.

3.3.3 Die Android-Bibliothek

Unter dem *gen*-Ordner folgt ein Ordner, der den Namen der anvisierten Target-Plattform trägt (in unserem Beispiel `ANDROID 4.0.3`) und in dem die JAR-Datei der zugehörigen Android-Bibliothek untergebracht ist.

Aufsteiger Sie möchten noch andere Java-Bibliotheken benutzen – vielleicht eine Geometrie-Bibliothek mit Klassen zum Durchführen von Koordinatentransformationen? Dann kopieren Sie die JAR-Dateien der Bibliotheken in den *gen*-Ordner Ihres App-Projekts.

JAR-Dateien

JAR-Dateien sind komprimierte ZIP-Archive mit Java-Klassen (und Interfaces). Da die Klassen in dem Archiv mitsamt ihren Paketpfaden abgespeichert sind, muss man das Archiv nicht extrahieren, um die Klassen der Bibliothek für das Schreiben eigener Java-Anwendungen benutzen zu können. Java-Compiler und -Interpreter können die nötigen Informationen aus der JAR-Datei herausziehen.

3.3.4 assets

Dieser Ordner ist anfänglich leer. Wir werden ihn in späteren Apps zum Ablegen von Datensammlungen, beispielsweise in Form von XML-Dateien oder SQLite-Datenbanken, nutzen.

3.3.5 Die Ressourcendateien

Zwei Arten von Ressourcendateien haben Sie schon kennengelernt: *activity_hallosagen.xml* und *strings.xml*. Welche weiteren Dateien es gibt, wie diese auf Ordner verteilt werden und wie man auf die in den Ressourcendateien definierten Ressourcen zugreift, ist uns ein eigenes Kapitel – das 6. – wert.

3.3.6 Die Manifestdatei

Aufgabe der Manifestdatei ist es, Ihre App dem Android-System vorzustellen, das die App installieren und ausführen soll. Dazu gehört unter anderem, dass die Manifestdatei angibt,

- welche Activities und anderen Komponenten zur App gehören,
- wie der Paketname der App lautet,
- für welche Android-Version die App geschrieben wurde (Target-SDK) und unter welcher Android-Version die App gerade noch so ausgeführt werden kann (Min-SDK).

Wenn Sie im Package Explorer auf den Ordner der Manifestdatei doppelklicken und dann in der unteren Registerlaschenreihe des Editors ganz rechts auf *AndroidManifest.xml* klicken, sehen Sie alle diese Informationen übersichtlich als XML-Code präsentiert.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="de.carpelibrum.hallo"
  android:versionCode="1"
  android:versionName="1.0" >
```

Wo wird was gespeichert? – Dateitypen, die Sie kennen sollten

```
<uses-sdk
  android:minSdkVersion="14"
  android:targetSdkVersion="15" />

<application
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <activity
    android:name=".HalloSagenActivity"
    android:label="@string/title_activity_hallo_sagen" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />

      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
</application>

</manifest>
```

Listing 3.2: Die Manifestdatei der Hallo-App

Wer mit der XML-Syntax auf Kriegsfuß steht, einmal nicht weiß, für welche Angabe er welches XML-Element oder -Attribut setzen muss, oder ganz einfach sichergehen möchte, keinen syntaktischen Fehler einzubauen, kann die gewünschten Angaben auch über die speziellen Eingabemasken vornehmen, die uns das Android-Plugin zur Verfügung stellt.

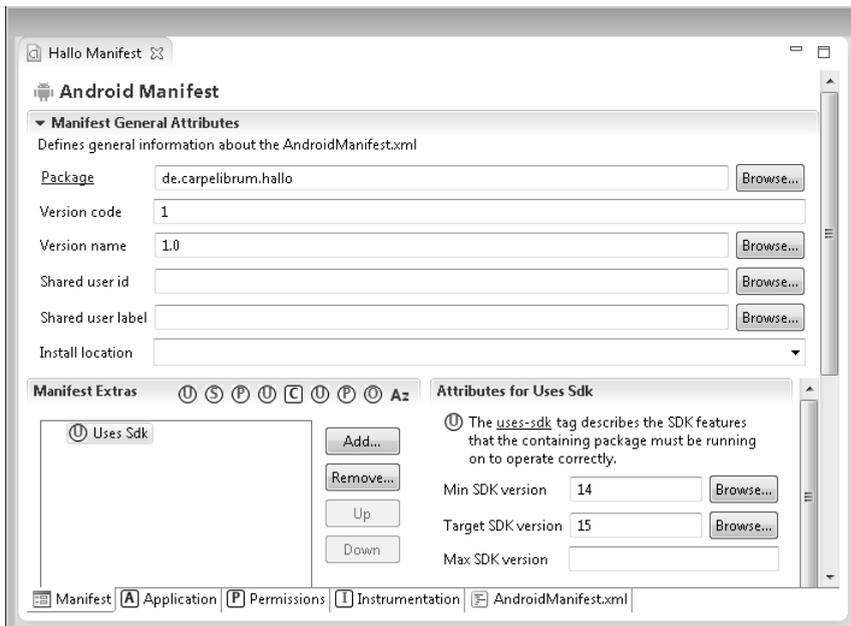


Abbildung 3.3: Festlegung des Target-SDK über die Eingabemaske Manifest

Die Start-Activity

Die Hallo-App, die wir in Kapitel 2 erstellt haben, besteht aus nur einer einzigen App-Komponente – der Activity `HalloSagenActivity` –, sodass es unter dem XML-Element `application` der Manifestdatei auch nur ein untergeordnetes Element gibt: eben für die Activity `HalloSagenActivity`:

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".HalloSagenActivity"
        android:label="@string/title_activity_hallo_sagen" >
        ...
    </activity>
</application>
```

Dies war zu erwarten¹, denn die Manifestdatei soll ja die Komponenten auflisten, die zur App gehören. Spannender ist da zweifelsohne das Unterelement `intent-filter`:

```
<activity
    android:name=".HalloSagenActivity"
    android:label="@string/title_activity_hallo_sagen" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

Intent-Filter dienen dazu, dem Android-System Sinn und Zweck einer Activity zu beschreiben. Der obige Intent-Filter teilt dem Android-System z.B. mit, dass die zugehörige App in der Liste der installierten Apps aufgeführt werden soll, von wo aus der Anwender sie starten kann (LAUNCHER-Kategorie), und dass diese Activity ausgeführt werden soll, wenn die App gestartet wird (MAIN-Action).

3.3.7 Die Properties-Datei

Die PROPERTIES-Datei gehört zum Android-Projekt und sollte nicht manuell bearbeitet werden. Wenn Sie die Projekteigenschaften bearbeiten möchten, klicken Sie stattdessen im Package Explorer mit der rechten Maustaste auf den Projektknoten und rufen Sie den Befehl PROPERTIES auf.

¹ Wenn Sie allerdings später zusätzliche Activities für eine App anlegen, müssen Sie diese eigenhändig in die Manifestdatei eintragen.

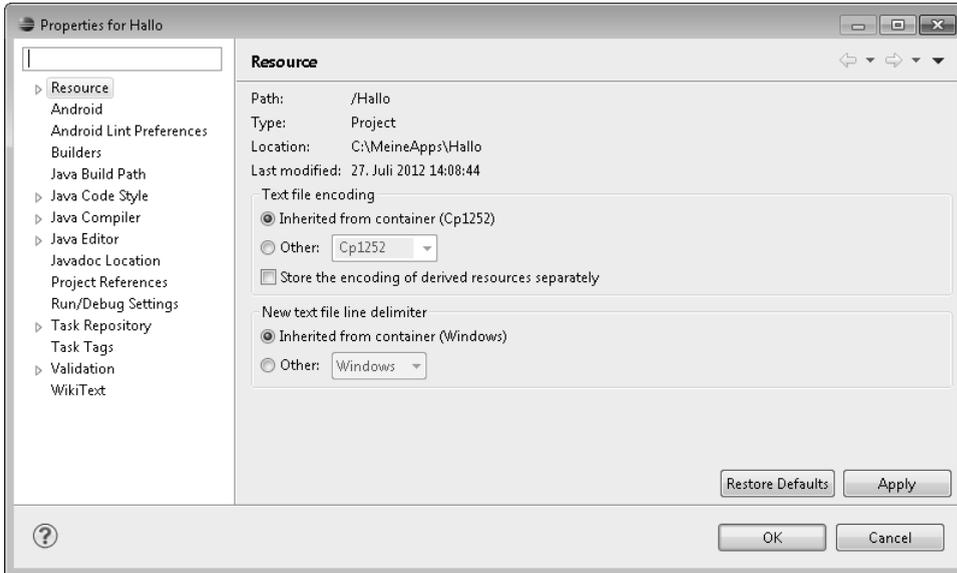


Abbildung 3.4: Die Projekteigenschaften bearbeiten

3.3.8 Die APK-Datei

Die APK-Datei wird durch Aufruf des Befehls `EXPORT` aus dem Kontextmenü des Projektknotens erzeugt. Sie ist die Installationsdatei der App und wird benötigt, wenn Sie Ihre App im Internet unter Google Play anbieten oder an Bekannte weitergeben möchten (siehe Anhang A).

3.4 Fragen und Antworten

1. Warum gibt es unter dem `res`-Verzeichnis so viele verschiedene `drawable`- und `values`-Ordner in meinem Projekt?

Eigentlich wollen wir uns mit dem Thema Ressourcen erst in Kapitel 6 näher befassen, aber da Sie schon fragen: Ressourcen, die von einer App verwendet werden sollen, müssen in speziellen Unterverzeichnissen des `res`-Ordners abgelegt werden. Bilddateien gehören z.B. in den Unterordner `drawable`. Nun ist es aber so, dass die App später auf Android-Geräte mit ganz unterschiedlicher Hardware oder Software-Konfigurationen ausgeführt wird. Android erlaubt daher dem App-Entwickler jede Ressource in mehreren Varianten anzubieten, die dann für spezielle Gerätekonfigurationen optimiert sind. Diese Varianten müssen dann in speziellen Verzeichnissen abgelegt werden, für Bilddateien in unterschiedlicher Auflösung beispielsweise `drawable-ldpi`, `drawable-mdpi` etc. Für den Moment soll dies als Erklärung genügen, ausführliche Informationen zur Bereitstellung individueller Ressourcen finden Sie dann in Kapitel 6.3.

Übrigens, welche `res`-Ordner für neue Projekte tatsächlich angelegt werden, hängt stark von dem verwendeten Android-Plugin ab.

- In der Manifestdatei habe ich eine Einstellung für die Max SDK-Version gesehen? Welche Funktion hat sie?

Über die Max SDK-Version können Sie verhindern, dass Ihre App auf Android-Geräten ab einer bestimmten API-Version installiert oder ausgeführt wird. Üblicherweise lässt man dieses Feld leer.

3.5 Übungen

- Erstellen Sie ein neues Projekt mit dem Namen *TitelDemoProjekt* für eine neue App namens *Titel-Demo* mit einer Activity namens *TitelActivity*, als deren Titel (TITLE-Option auf der Dialogseite NEW BLANK ACTIVITY) Sie »Der Titel« eingeben. Füllen Sie dann mithilfe dieser App die folgende Tabelle aus, anhand deren man erkennen soll, wo und wie die Projektvorgaben in den Quelldateien gespeichert werden und wo diese Werte dann später im Emulator und der ausgeführten App auftauchen.

Projektvorgabe App-Wizard	Code	Emulator/App
APPLICATION NAME		
PROJECT NAME	–	–
ACTIVITY NAME		–
TITLE		

Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt. Dieses eBook stellen wir lediglich als **persönliche Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschließlich

- der Reproduktion,
- der Weitergabe,
- des Weitervertriebs,
- der Platzierung im Internet, in Intranets, in Extranets,
- der Veränderung,
- des Weiterverkaufs und
- der Veröffentlichung

bedarf der **schriftlichen Genehmigung** des Verlags. Insbesondere ist die Entfernung oder Änderung des vom Verlag vergebenen Passwortschutzes ausdrücklich untersagt!

Bei Fragen zu diesem Thema wenden Sie sich bitte an: info@pearson.de

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf unseren Websites ist eine freiwillige Leistung des Verlags. **Der Rechtsweg ist ausgeschlossen.**

Hinweis

Dieses und viele weitere eBooks können Sie rund um die Uhr und legal auf unserer Website herunterladen:

<http://ebooks.pearson.de>