jetzt lerne ich

Android

Der schnelle Einstieg in die Programmierung und Entwicklungsumgebung

Markt+Technik

DIRK LOUIS / PETER MÜLLER

- Aufbau grafischer Benutzeroberflächen
- Mehrere vollständige Apps zunehmender Komplexität
- Bedienungsanleitung zur Eclipse-Entwicklungsumgebung
- Java-Tutorium für Umsteiger von anderen Programmiersprachen

ALWAYS LEARNING

PEARSON

Jetzt lerne ich Android

jetzt lerne ich

Android

Der schnelle Einstieg in die Programmierung und Entwicklungsumgebung

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

10 9 8 7 6 5 4 3 2 1
13 12 11

ISBN 978-3-8272-4715-5

ein Imprint der Pearson Deutschland GmbH,
Martin-Kollar-Straße 10-12, D-81829 München/Germany

Covergestaltung: Thomas Arlt, tarlt@adesso21.net
Lektorat: Brigitte Bauer-Schiewek, bbauer@pearson.de
Fachlektorat: Petra Alm
Herstellung: Martha Kürzl-Harrison, mkuerzl@pearson.de
Korrektorat: Petra Alm
Satz: text&form GbR, Fürstenfeldbruck
Druck und Verarbeitung: Drukarnia Dimograf, Bielsko-Biala
Printed in Poland

3 Was wann wofür

Sie lernen in diesem Kapitel,

- *wie Apps aufgebaut sind,*
- *welche Bibliotheksklassen 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-CD 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 etlichen 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 usw.), 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

Für jede Bildschirmseite eine eigene Activity

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.

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 vorneherein mitbringt: die einfache anwendungsübergreifende Wiederverwendung von auf dem System installierten Software-Bausteinen.

Intents

Activities kommunizieren über Intents

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

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.)

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.

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.

Views

Bildschirmseiten werden aus Views aufgebaut

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*).

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.

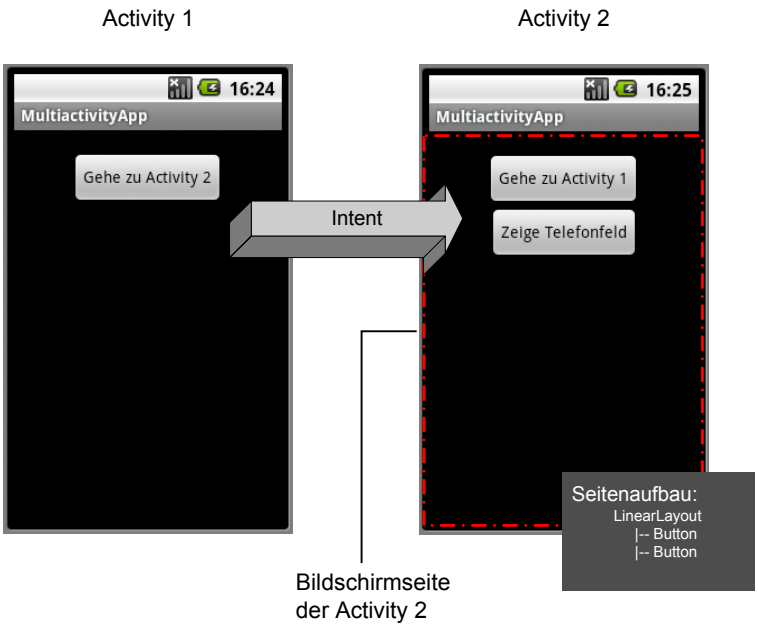


Abbildung 3.1:
Drückt der Anwender den Button auf der 1. Bildschirmseite, wird ein Intent abgesetzt, der die Bildschirmseite der 2. Activity aufruft. Will der Anwender zurück zur 1. Seite, muss er auf den zugehörigen Button drücken, der natürlich ebenfalls einen Intent absetzt (hier nicht dargestellt).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

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.

- Fragments

  Mit Android 3.0 (API-Level 11) eingeführte Komponente. Fragments erlauben dem Programmierer, den Code umfangreicher Activities aufzuteilen, indem er Teile der Activity in Fragments auslagert. Fragments definieren ihre eigene Benutzeroberfläche und haben einen eigenen Lebenszyklus.

App-Komponenten

Activities, Services, Content Provider, Broadcast Receiver und Fragments 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.

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

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*).

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

Tabelle 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.

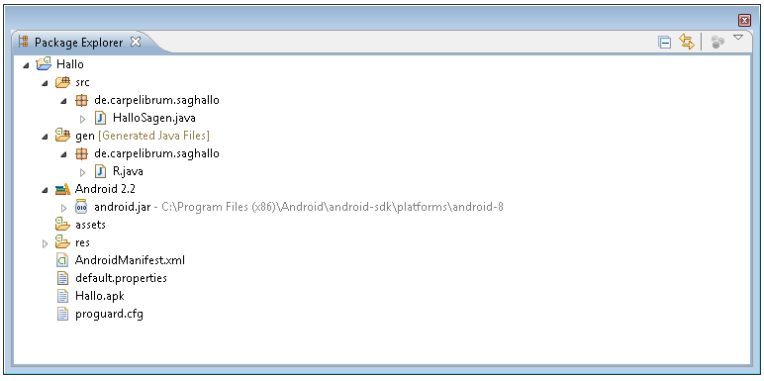


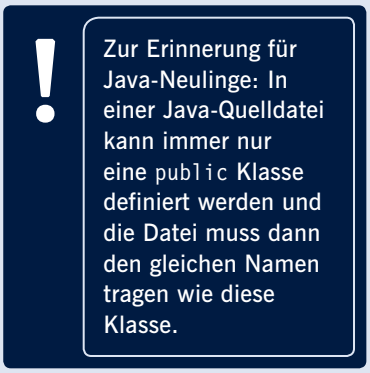
Abbildung 3.2:
Aufbau eines Android-Projekts (dargestellt im Package Explorer von Eclipse)

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 Startactivity 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*).

Die Quelldateien der App müssen – wir haben es bereits mehrfach angesprochen – in einem eigenen Paket (hier `de.carpelibrum.saghallo`) 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.



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

R.java wird im Zuge der Projekterstellung aktualisiert

Im Ordner *gen* steht 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.

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

```
/* 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.arpelibrum.saghallo;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
```

```
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

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

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

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

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

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

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 2.2) und in dem die JAR-Datei der zugehörigen Android-Bibliothek untergebracht ist.

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.

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.

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

3.3.5 Die Ressourcendateien

Zwei Arten von Ressourcendateien haben Sie schon kennengelernt: *main.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,
- welche Android-Version die App anvisiert (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.

Listing 3.2: Die Manifestdatei der Hallo-App

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="de.carpelibrum.saghallo"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8"
              android:targetSdkVersion="8"/>

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        <activity android:name=".HalloSagen"
                  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>
```

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.

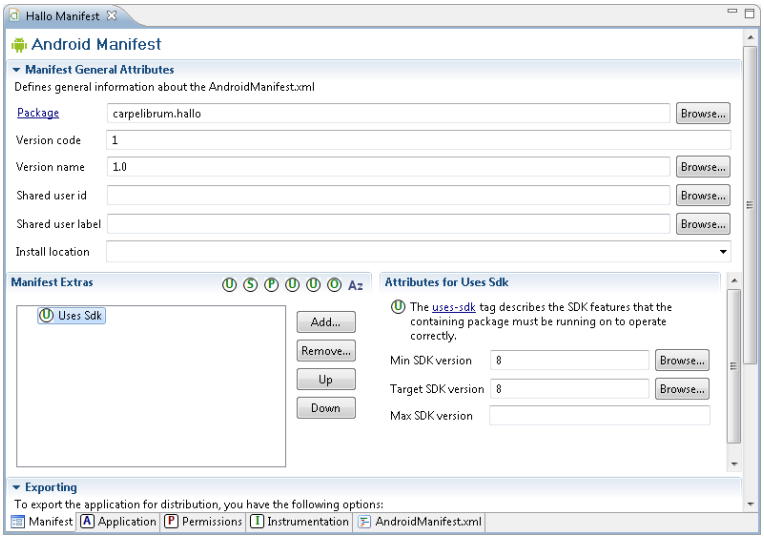


Abbildung 3.3:
Festlegung des Target-SDK über die Eingabemaske Manifest

Die Start-Activity

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

```
<application android:icon="@drawable/icon"
             android:label="@string/app_name">
    <activity android:name=".HalloSagen"
              android:label="@string/app_name">
        ...
    </activity>
</application >
```

Dies war zu erwarten[1], denn die Manifestdatei soll ja die Komponenten auflisten, die zur App gehören. Spannender ist da zweifelsohne das Unterelement `intent-filter`:

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

```
<activity android:name=".HalloSagen"
          android:label="@string/app_name">
     <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.

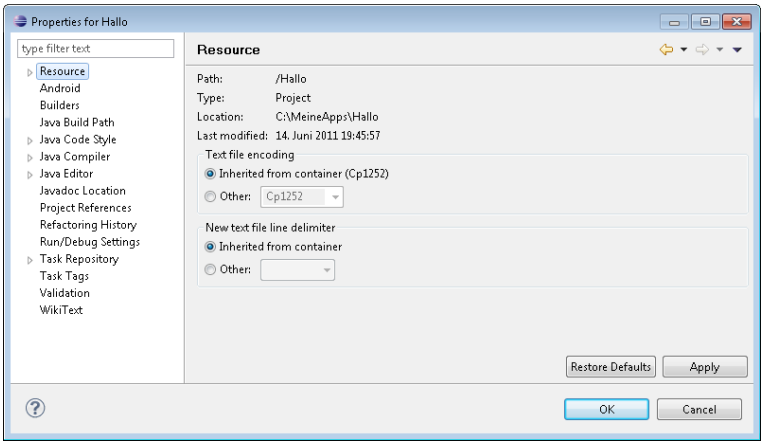


Abbildung 3.4: Die Projekteigenschaften bearbeiten

3.3.8 Die APK-Datei

Die APK-Datei ist sozusagen die Installationsdatei der App und wird über den Befehl EXPORT aus dem Kontextmenü des Projektknotens erzeugt (siehe *Kapitel 2.7*).

Copyright