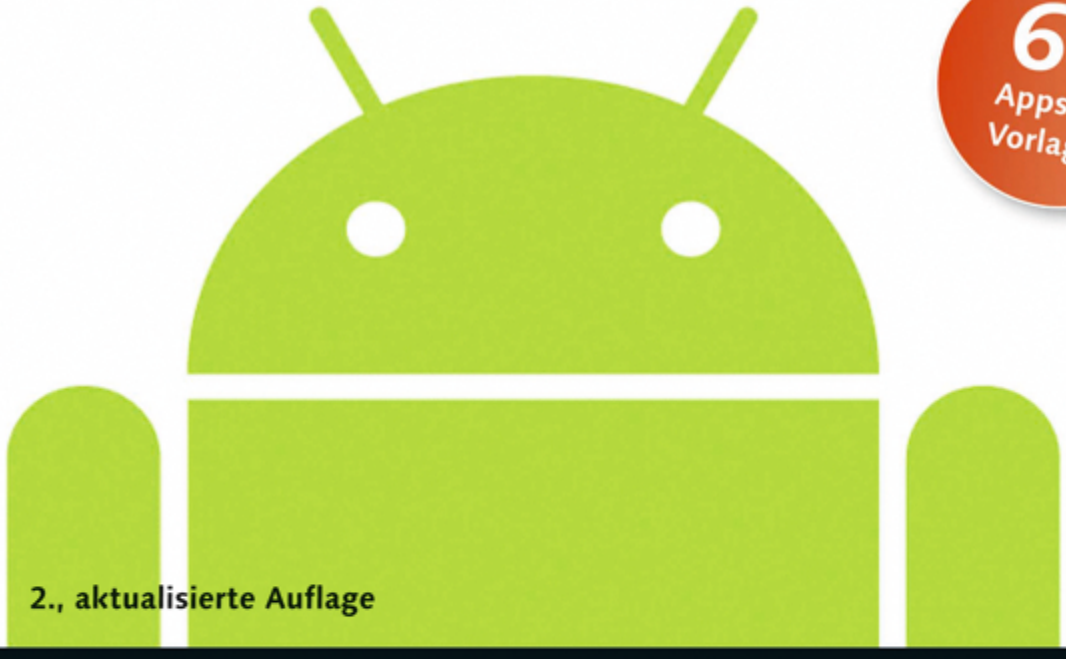
Thomas Künneth

Android 4

Apps entwickeln mit dem Android SDK

60 Apps als Vorlagen



2., aktualisierte Auflage

- Apps für Smartphones und Tablets entwickeln
- Von der Idee bis zur Veröffentlichung
- Inkl. Multimedia, GPS, Kalender, GUIs, Multitasking u. v. m.

LEHR-Programm gemäß §14 JuSchG

DVD ROM

60 Beispiel-Apps, alle Entwicklungstools, Video-Training »Eine Notizzettel-App programmieren«, Java-Kompendium

Galileo Computing

Thomas Künneth

Android 4

Apps entwickeln mit dem Android SDK

Galileo Press

Liebe Leserin, lieber Leser,

mit der Entscheidung, Apps für Android zu entwickeln, haben Sie eine ausgezeichnete Wahl getroffen, denn Android boomt. Das liegt nicht zuletzt daran, dass Google es den Entwicklern leicht macht: Programmiert wird mit einer weit verbreiteten Sprache wie Java, es stehen kostenlose, leistungsfähige Entwicklungswerkzeuge zur Verfügung, und die Zugangshürden für Google Play sind niedrig.

Die Ausgangsvoraussetzungen könnten besser nicht sein – höchste Zeit also, eigene Android-Apps zu entwickeln! Um Sie dabei optimal zu unterstützen, lernen Sie in diesem Buch zunächst die Entwicklungswerkzeuge und den grundlegenden Aufbau von Apps kennen und erfahren, wie Sie Ihre fertigen Apps veröffentlichen können.

Im weiteren Verlauf zeigt Ihnen der erfahrene Android- und Java-Entwickler Thomas Künneth dann, wie Sie die Möglichkeiten von Android voll ausreizen: Sie lernen, wie Sie in Ihren Apps Anrufe entgegennehmen, die Audio- und Videofunktionen nutzen, GPS und Sensoren ansprechen, auf Termine und Kontakte zugreifen u. v. m. Zu allen Themen gibt es anschauliche Beispiel-Apps, die Sie auch auf der beiliegenden DVD finden und als Ausgangsbasis für eigene Apps nutzen können.

Dabei spielt es keine Rolle, ob Sie für Smartphones oder für Tablets entwickeln möchten – die aktuelle Android-Version 4 ist für beide Systeme geeignet. Wenn Sie grundlegende Java-Kenntnisse mitbringen, steht Ihren eigenen Android-Apps nichts mehr im Wege!

Dieses Buch wurde mit großer Sorgfalt geschrieben, geprüft und produziert. Sollte dennoch einmal etwas nicht so funktionieren, wie Sie es erwarten, freue ich mich, wenn Sie sich mit mir in Verbindung setzen. Ihre Kritik und konstruktiven Anregungen sind uns jederzeit herzlich willkommen!

Viel Spaß beim Entwickeln Ihrer Android-Apps wünscht Ihnen

Ihre Judith Stevens-Lemoine
Lektorat Galileo Computing

judith.stevens@galileo-press.de
www.galileocomputing.de
Galileo Press · Rheinwerkallee 4 · 53227 Bonn

Auf einen Blick

TEIL I Grundlagen

1 Android – eine offene, mobile Plattform 21

2 Hallo Android! 39

3 Von der Idee zur Veröffentlichung 61

TEIL II Elementare Anwendungsbausteine

4 Activities und Broadcast Receiver 87

5 Benutzeroberflächen 117

6 Multitasking 159

TEIL III Telefonfunktionen nutzen

7 Rund ums Telefonieren 185

8 Widgets und Wallpapers 195

9 Sensoren und GPS 227

TEIL IV Dateien und Datenbanken

10 Das Android-Dateisystem 251

11 Datenbanken 269

12 Content Provider 289

13 Daten sichern und wiederherstellen 309

TEIL V Organizer und Multimedia

14 Audio 335

15 Fotos und Video 367

16 Kontakte und Organizer 397

Der Name Galileo Press geht auf den italienischen Mathematiker und Philosophen Galileo Galilei (1564 – 1642) zurück. Er gilt als Gründungsfigur der neuzeitlichen Wissenschaft und wurde berühmt als Verfechter des modernen, heliozentrischen Weltbilds. Legendär ist sein Ausspruch *Eppur si muove* (Und sie bewegt sich doch). Das Emblem von Galileo Press ist der Jupiter, umkreist von den vier Galileischen Monden. Galilei entdeckte die nach ihm benannten Monde 1610.

Lektorat Judith Stevens-Lemoine, Anne Scheibe
Korrektorat Stefan Mann, Köln
Einbandgestaltung Barbara Thoben, Köln
Titelbild Google Inc.
Typografie und Layout Vera Brauner
Herstellung Katrin Müller
Satz Typographie & Computer, Krefeld
Druck und Bindung Beltz Druckpartner, Hemsbach

Gerne stehen wir Ihnen mit Rat und Tat zur Seite

judith.stevens@galileo-press.de
bei Fragen und Anmerkungen zum Inhalt des Buches

service@galileo-press.de
für versandkostenfreie Bestellungen und Reklamationen

britta.behrens@galileo-press.de
für Rezensions- und Schulungsexemplare

Dieses Buch wurde gesetzt aus der TheAntiquaB (9,35/13,7 pt) in FrameMaker.

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.

ISBN 978-3-8362-1948-8

2., aktualisierte und erweiterte Auflage 2012

Für Moni

Inhalt

Vorwort 15

TEIL I Grundlagen

1 Android – eine offene, mobile Plattform 21

1.1 Entstehung 21
1.1.1 Die Open Handset Alliance 22
1.1.2 Android Inc. 22
1.1.3 Evolution einer Plattform 23
1.2 Systemarchitektur 25
1.2.1 Überblick 25
1.2.2 Android Runtime 27
1.2.3 Application Framework 28
1.3 Entwicklungswerkzeuge 29
1.3.1 Android SDK 30
1.3.2 Android Development Tools 34
1.3.3 Das erste eigene Projekt 35
1.3.4 Android-Quelltexte einhängen 36

2 Hallo Android! 39

2.1 Android-Projekte 39
2.1.1 Projekte anlegen 40
2.1.2 Projektstruktur 41
2.1.3 Run Configurations 44
2.2 Die Benutzeroberfläche 47
2.2.1 Texte 48
2.2.2 Views 51
2.2.3 Oberflächenbeschreibungen 52
2.3 Programmlogik und -ablauf 54
2.3.1 Activities 54
2.3.2 Benutzereingaben 57
2.3.3 Der letzte Schliff 58

3 Von der Idee zur Veröffentlichung 61

3.1 Konzept und Realisierung ... 61
3.1.1 Konzeption ... 62
3.1.2 Fachlogik ... 63
3.1.3 Benutzeroberfläche ... 66

3.2 Vom Programm zum Produkt ... 71
3.2.1 Protokollierung ... 72
3.2.2 Fehler suchen und finden ... 75
3.2.3 Debuggen auf echter Hardware ... 77

3.3 Anwendungen verteilen ... 79
3.3.1 Verteilbare Anwendungen ... 79
3.3.2 Apps in Google Play einstellen ... 81
3.3.3 Alternative Märkte und Ad hoc-Verteilung ... 83

TEIL II Elementare Anwendungsbausteine

4 Activities und Broadcast Receiver 87

4.1 Was sind Activities? ... 87
4.1.1 Struktur von Apps ... 87
4.1.2 Lebenszyklus von Activities ... 94

4.2 Kommunikation zwischen Anwendungsbausteinen ... 101
4.2.1 Intents ... 102
4.2.2 Kommunikation zwischen Activities ... 104
4.2.3 Broadcast Receiver ... 106

4.3 Fragmente ... 109
4.3.1 Grundlagen ... 109
4.3.2 Ein Fragment in eine Activity einbetten ... 112

5 Benutzeroberflächen 117

5.1 Views und ViewGroups ... 117
5.1.1 Views ... 118
5.1.2 Positionierung von Bedienelementen mit ViewGroups ... 124

5.2 Alternative Ressourcen ... 130
5.2.1 Automatische Layout-Auswahl ... 130
5.2.2 Bitmaps und Pixeldichte ... 136
5.3 Vorgefertigte Bausteine für Oberflächen ... 137
5.3.1 Nützliche Activities ... 137
5.3.2 Dialoge ... 144
5.3.3 Menüs ... 148
5.3.4 Action Bar ... 153

6 Multitasking 159

6.1 Threads ... 160
6.1.1 Threads in Java ... 160
6.1.2 Vom Umgang mit Threads in Android ... 165
6.2 Services ... 169
6.2.1 Gestartete Services ... 169
6.2.2 Gebundene Services ... 174

TEIL III Telefonfunktionen nutzen

7 Rund ums Telefonieren 185

7.1 Telefonieren ... 185
7.1.1 Anrufe tätigen ... 185
7.1.2 Auf eingehende Anrufe reagieren ... 187
7.2 Telefon- und Netzstatus ... 189
7.2.1 Geräte identifizieren ... 189
7.2.2 Netzwerkinformationen anzeigen ... 190
7.3 Das Call Log ... 190
7.3.1 Entgangene Anrufe ... 191
7.3.2 Einträge bearbeiten ... 192
7.3.3 Benachrichtigung bei Änderungen ... 193

8 Widgets und Wallpapers 195

8.1 Widgets ... 195
8.1.1 Beteiligte Klassen und Dateien ... 195
8.1.2 Die Benutzeroberfläche ... 202
8.1.3 Vorschau-Bilder ... 206

8.2 Wallpaper ... 208
8.2.1 Die Wallpaper-API ... 208
8.2.2 Hintergründe auswählen ... 209

8.3 Live Wallpaper ... 214
8.3.1 WallpaperService und Engine ... 214
8.3.2 Live Wallpaper auswählen ... 219
8.3.3 Einstellungsseiten ... 222

9 Sensoren und GPS 227

9.1 Sensoren ... 227
9.1.1 Die Klasse SensorManager ... 227
9.1.2 Sensoren simulieren ... 230

9.2 GPS und ortsbezogene Dienste ... 237
9.2.1 Den aktuellen Standort ermitteln ... 237
9.2.2 Positionen in einer Karte anzeigen ... 242

TEIL IV Dateien und Datenbanken

10 Das Android-Dateisystem 251

10.1 Grundlegende Dateioperationen ... 251
10.1.1 Dateien lesen und schreiben ... 252
10.1.2 Mit Verzeichnissen arbeiten ... 257

10.2 Externe Speichermedien ... 261
10.2.1 Mit SD-Cards arbeiten ... 261
10.2.2 Installationsort von Apps ... 265

11 Datenbanken 269

11.1 Erste Schritte mit SQLite ... 269
11.1.1 Was ist SQLite? ... 270
11.1.2 Auf der Kommandozeile arbeiten ... 271
11.1.3 SQLite in Apps nutzen ... 274

11.2 Fortgeschrittene Operationen ... 280
11.2.1 Klickverlauf mit SELECT ermitteln ... 281
11.2.2 Daten mit UPDATE ändern und mit DELETE löschen ... 286

12 Content Provider 289

12.1 Vorhandene Content Provider nutzen ... 290
12.1.1 Mit Content Resolver auf Wörterbücher zugreifen ... 290
12.1.2 Browser-Bookmarks ... 294

12.2 Implementierung eines eigenen Content Providers ... 297
12.2.1 Anpassungen an der App TKMoodley ... 297
12.2.2 Die Klasse android.content.ContentProvider ... 302

13 Daten sichern und wiederherstellen 309

13.1 Grundlagen ... 309
13.1.1 Beteiligte Komponenten ... 310
13.1.2 BackupAgentHelper ... 313
13.1.3 Den Backup Agent testen ... 316

13.2 Eigene Dateien und Datenobjekte sichern ... 319
13.2.1 FileBackupHelper ... 319
13.2.2 Von BackupAgent ableiten ... 323

TEIL V Organizer und Multimedia

14 Audio 335

14.1 Rasender Reporter – ein Diktiergerät als App 335
14.1.1 Struktur der App 335
14.1.2 Audio aufnehmen und abspielen 338

14.2 Effekte 344
14.2.1 Die Klasse AudioEffekteDemo 344
14.2.2 Bass Boost und Virtualizer 347
14.2.3 Hall 349

14.3 Sprachsynthese 350
14.3.1 Nutzung der Sprachsynthesekomponente vorbereiten 351
14.3.2 Texte vorlesen 356
14.3.3 Sprachausgaben speichern 358

14.4 Weitere Audiofunktionen 359
14.4.1 Spracherkennung 359
14.4.2 Tastendrücke von Headsets verarbeiten 363

15 Fotos und Video 367

15.1 Vorhandene Activities nutzen 367
15.1.1 Kamera-Activity starten 367
15.1.2 Aufgenommene Fotos weiterverarbeiten 370
15.1.3 Mit der Galerie arbeiten 374
15.1.4 Die Kamera-App erweitern 376

15.2 Die eigene Kamera-App 381
15.2.1 Live-Vorschau 381
15.2.2 Kamera auswählen 386
15.2.3 Fotos aufnehmen 387

15.3 Videos drehen 390
15.3.1 Die App VideoCaptureDemo 390
15.3.2 MediaRecorder konfigurieren 394

16 Kontakte und Organizer 397

16.1 Kontakte ... 397
16.1.1 Eine einfache Kontaktliste ausgeben ... 397
16.1.2 Weitere Kontaktdaten ausgeben ... 399
16.1.3 Geburtstage hinzufügen und aktualisieren ... 402

16.2 Auf Google-Konten zugreifen ... 408
16.2.1 Emulator konfigurieren ... 409
16.2.2 Aufgabenliste auslesen ... 412

16.3 Kalender und Termine ... 417
16.3.1 Termine anlegen und auslesen ... 417
16.3.2 Alarme verwalten ... 419
16.3.3 Die Klasse CalendarContract ... 423

Anhang 425

A Literaturverzeichnis ... 427

B Die Begleit-DVD ... 429

C Häufig benötigte Code-Bausteine ... 435
C.1 Manifestdatei ... 435
C.2 Berechtigungen ... 437
C.2.1 Hardware, Telefonie und Netzwerk ... 437
C.2.2 Internet ... 437
C.2.3 Audio und Video ... 438
C.2.4 Kontakte und Kalender ... 438

Index ... 439

Vorwort

Als die Deutsche Telekom Anfang 2009 das Google G1 vorstellte, war die Neugier groß. Ein Handy des Suchmaschinenprimus ließ auf eine enge Integration seiner Dienste und damit auf viele spannende, neue Möglichkeiten hoffen. Dass das erste Android-Smartphone die hoch gesteckten Erwartungen nicht erfüllen konnte, darf angesichts eines scheinbar übermächtigen Konkurrenten nicht verwundern. Apple hatte seit der Einführung des iPhone akribisch und zielstrebig an seiner Plattform und den sie umgebenden Produkten gearbeitet. Das Ergebnis war ein Ökosystem, gegen das der Neuling aus Mountain View zum damaligen Zeitpunkt im direkten Vergleich keine Chance hatte.

Seitdem ist viel passiert. Android hat sich zum Liebling von Verbrauchern, Entwicklern und Herstellern gemausert. Die Flut an neuen Modellen nimmt kein Ende. Und Apple, Google und der wieder erstarkte Mitbewerber Microsoft liefern sich ein Wettrennen um die besten Innovationen. Den Nutzern von mobilen Geräten kann diese Situation nur recht sein, und letztlich profitieren alle Lager von diesem Wettstreit. Die Frage, warum Android mittlerweile so populär ist, lässt sich schnell beantworten: Google hat von Anfang an auf Offenheit gesetzt. Jeder war und ist eingeladen mitzumachen. Hardware-Hersteller können Produkte entwickeln, ohne hohe Summen an Lizenzkosten zahlen zu müssen. Und interessierten Programmierern steht mit dem Android Software Development Kit und den Android Development Tools ein leistungsfähiges Gespann zur Entwicklung von Apps zur Verfügung.

Dank Java als Programmiersprache und Eclipse als Entwicklungsumgebung fällt der Einstieg im Vergleich zu manch anderer Plattform leicht. Dennoch gibt es für Einsteiger in diese faszinierende Welt vieles zu beachten. Android bietet schier unendliche Möglichkeiten. Und mit jeder Plattform-Version kommen neue Funktionen hinzu. Um diese sicher nutzen zu können, müssen Sie als Entwickler mit einer Reihe von Mechanismen und Konzepten vertraut sein. Dieses Wissen möchte ich Ihnen mit dem vorliegenden Buch gerne vermitteln. Aber nicht in Form einer theoretischen Abhandlung. In vielen kleinen, in sich geschlossenen praxisnahen Beispielen lernen Sie die souveräne Nutzung der Android-Programmierschnittstellen kennen. Natürlich finden Sie alle Programme als Eclipse-Projekte auch auf der Begleit-DVD.

Aufbau des Buchs

Das Buch ist in fünf Teile gegliedert. In Teil 1, »Grundlagen«, stelle ich Ihnen Android und seine Entwicklerwerkzeuge vor und begleite Sie Schritt für Schritt zu Ihrer ersten App. Außerdem lernen Sie Google Play als moderne digitale Vertriebsplattform kennen. Der zweite Teil, »Elementare Anwendungsbausteine«, beschäftigt sich mit Komponenten, die in nahezu jeder App vorhanden sind. Hierzu gehört natürlich die

Benutzeroberfläche. Aber auch Multitasking und wie es in Android umgesetzt wird, beschreibe ich in diesem Teil ausführlich. In »Telefonfunktionen nutzen« erstellen Sie unter anderem eigene Widgets und Wallpaper und lernen die Sensoren eines Android-Geräts kennen. »Dateien und Datenbanken« befasst sich nicht nur mit der Speicherung und Abfrage von Daten, ich zeige Ihnen außerdem, wie Sie Ihre Apps fit für die Installation auf Wechselmedien machen und Googles Sicherungsdienst nutzen, um Anwendungsdaten in der Cloud zu speichern. Im fünften Teil »Organizer und Multimedia« schließlich nutzen Sie das Mikrofon eines Android-Geräts, um Geräusche aufzunehmen. Sie schießen Fotos mit der eingebauten Kamera und legen Kontakte und Termine an.

Jedes Kapitel ist in sich abgeschlossen und beschäftigt sich mit genau einem Themenkomplex. Wenn Sie schon etwas Erfahrung mit Android haben, müssen Sie das Buch also nicht von Deckel zu Deckel durcharbeiten, sondern können sich gezielt einen Aspekt herausgreifen, der Sie besonders interessiert. Neulingen möchte ich die beiden ersten Kapitel als Einstieg ans Herz legen. Sie lernen die Bestandteile von Googles offener Plattform für mobile Geräte kennen, installieren alle benötigten Komponenten und schreiben dann Ihre erste App.

Programmierkenntnisse

Um die Beispiele nachvollziehen zu können, müssen Sie kein Java-Profi sein, allerdings sollten Sie diese Programmiersprache und ihre Klassenbibliothek zumindest in Grundzügen beherrschen. In Anhang A finden Sie eine Literaturliste mit empfehlenswerten Büchern für den Einstieg in Java. Außerdem finden Sie das Kompendium *Java ist auch eine Insel* von Christian Ullenboom auf der Begleit-DVD.

Unterstützte Android-Versionen

Dieses Buch beleuchtet die Anwendungsentwicklung unter Android 4 (Ice Cream Sandwich). Allerdings ist es nicht mein Ziel, »einfach nur« alle Neuerungen dieser Systemversion aufzulisten, sondern Ihnen zu zeigen, wie Sie Anwendungen programmieren, die auf möglichst vielen aktuellen Geräten funktionieren. Im Gegensatz zu anderen Plattformen dauert es nämlich recht lange, bis Hersteller ihre Produkte mit Updates versorgen. Nicht wenige Modelle gehen gänzlich leer aus. Neue Plattform-Versionen nehmen also nur sehr langsam Fahrt auf. Um einen möglichst breiten Kundenkreis zu erreichen, sollte Ihre App also nicht nur auf ganz neuen Systemen funktionieren. Viele der vorgestellten Konzepte und Mechanismen waren deshalb schon in früheren Plattformen vorhanden. Aber natürlich stelle ich Ihnen auch etliche echte Neuerungen vor, unter anderem die Kalender-API. Wichtig: Sie können meine Beispielprogramme für Smartphones und Tablets gleichermaßen nutzen.

Danksagung

Dieses Buch wäre ohne die Unterstützung von vielen Menschen nicht möglich gewesen. Ihnen allen gebührt mein tief empfundener Dank. Dazu gehören die Mitarbeiterinnen und Mitarbeiter des Verlags Galileo Press, insbesondere meine Lektorin Judith Stevens-Lemoine und Frau Anne Scheibe. Der Weg von der Idee über das Manuskript bis zum fertigen Buch ist lang und manchmal steiniger als erwartet. Für stets offene Ohren, freundliche wie professionelle Unterstützung und manchmal auch Geduld bedanke ich mich herzlich.

Mit bewundernswerter Akribie haben Yvonne Wolf, Dietz Pröpper, Thomas Bednarek, Mathias Hengl und Dave Richardson jedes einzelne Kapitel gelesen und korrigiert und somit geholfen, die eine oder andere Ungereimtheit zu klären sowie manche Kante zu glätten. Hierfür und für zahlreiche wertvolle Tipps vielen Dank.

Meinen Eltern Rudolf und Gertraud Künneth und meinem Bruder Andreas Künneth danke ich für alles, was sie mir auf den Weg gegeben haben. Ohne sie wäre vieles nicht möglich. Der allergrößte Dank aber gebührt meiner Ehefrau Moni für das unermessliche Glück, das sie mir jeden Tag schenkt, für ihre Liebe, ihre Unterstützung und Geduld. Ihr widme ich dieses Buch.

TEIL I

Grundlagen

Kapitel 1
Android – eine offene, mobile Plattform

Was genau ist Android eigentlich? Wie ist die Plattform entstanden? Und aus welchen Bausteinen und Schichten besteht sie? Dieses Kapitel macht Sie mit wichtigen Grundlagen vertraut.

Die Anwendungsentwicklung für Android macht – Sie werden mir nach der Lektüre dieses Buches sicherlich zustimmen – großen Spaß. Zum einen, weil diese Plattform unglaublich viele Möglichkeiten bietet. Unzählige Programmierschnittstellen und Funktionen warten darauf, erkundet und genutzt zu werden. Zum anderen ist die Entwicklungsumgebung, also der Werkzeugkasten des Programmierers, äußerst komfortabel. Routinetätigkeiten gehen deshalb reibungslos und glatt von der Hand.

Allerdings müssen Sie als Entwickler die angebotenen Möglichkeiten natürlich auch zu nutzen verstehen. Dies betrifft nicht nur die Bedienung der Werkzeuge, sondern auch das Wissen um die Zusammenhänge zwischen den einzelnen Bausteinen und Schichten der Plattform. In diesem Kapitel zeige ich Ihnen deshalb unter anderem, wie Android aufgebaut ist und aus welchen Funktionsgruppen und Schichten das System besteht. Zunächst möchte ich Ihnen aber kurz die Entstehung der Plattform erläutern.

1.1 Entstehung

Am 12. November 2007 kündigte Google die Verfügbarkeit einer frühen Vorschauversion des Android Software Development Kits (SDK) an.[1] Entwickler konnten damit zum ersten Mal Programme für eine bis dahin völlig unbekannte Plattform schreiben und in einem Emulator ausprobieren. Das erste Gerät (das durch T-Mobile vertriebene G1) stand Kunden in Deutschland allerdings erst über ein Jahr später zur Verfügung. In dieser Zeit hatte Google das System zu einer ersten, halbwegs endanwendertauglichen Version 1.1 weiterentwickelt.

1 *http://android-developers.blogspot.com/2007/11/posted-by-jason-chen-android-advocate.html*

1.1.1 Die Open Handset Alliance

Genau eine Woche vor der Veröffentlichung der Android-SDK-Vorschau war die *Open Handset Alliance (OHA)* erstmals an die Öffentlichkeit getreten. Dieses durch den Suchmaschinenprimus angeführte Konsortium bestand damals aus 34 Firmen (unter anderem Halbleiter- und Mobiltelefonhersteller, Netzbetreiber und Softwareentwickler). Die Allianz hatte ihre Absicht verkündet, mit Android die erste wirklich offene Plattform für mobile Geräte zu schaffen. Die in der Pressemitteilung vom 5. November 2007[2] formulierten Ziele waren unter anderem:

- eine deutliche Verbesserung der Benutzerfreundlichkeit und des Benutzererlebnisses von mobilen Geräten und Diensten
- die kostengünstigere Entwicklung und Verteilung innovativer Produkte
- eine schnellere Markteinführung

Nach Ansicht der OHA ließ sich dies am besten durch eine auf Offenheit und Flexibilität gründende Zusammenarbeit von Entwicklern, Herstellern und Betreibern erreichen. Deshalb wurden praktisch alle Teile des Android-Softwarestapels als Open Source veröffentlicht. Zudem hatten Gerätebauer sowie Netzanbieter von Anfang an die Möglichkeit, die Plattform anzupassen oder zu erweitern. Auch Entwickler profitieren von diesem Ansatz, indem sie Alternativen zu Kernkomponenten (beispielsweise dem mitgelieferten Webbrowser) anbieten können. Dies war übrigens auf dem iPhone lange Zeit nicht möglich.

Während die Anzahl der Android-basierten Geräte 2009 noch recht überschaubar war, haben ein Jahr später zahlreiche Hersteller entsprechende Produkte angekündigt und auch ausgeliefert. Die kontinuierlich wachsende Bedeutung der Plattform spiegelt sich auch in der Mitgliederzahl der OHA wieder. Sie ist auf über 80 Firmen angewachsen.

1.1.2 Android Inc.

Die Pressemitteilung der OHA beendete zunächst Spekulationen der Medien, Google könne die Einführung eines eigenen Mobiltelefons planen. Dass sich der Suchmaschinenriese für den Markt um mobile Kommunikation stark interessiert, war schon in der Vergangenheit häufiger thematisiert worden.

Im Juli 2005 hatte Google ein kleines Startup-Unternehmen namens Android Inc. mit Sitz im kalifornischen Palo Alto übernommen.[3] Außer den Namen einiger Mitarbeiter war zu diesem Zeitpunkt sehr wenig über die Firma bekannt. Sie hatte stets im Verborgenen gearbeitet. Die Vermutung, man entwickle ein Betriebssystem für mobile Geräte, wurde auch nach dem Kauf nicht kommentiert. Die offizielle Sprach-

2 *www.openhandsetalliance.com/press_110507.html*

3 *www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm*

regelung war, Android Inc. sei wegen der talentierten Ingenieure sowie der von ihnen entwickelten Technologie übernommen worden.

In der Tat brachten die Silicon Valley-Veteranen Andy Rubin, Richard Miner, Nick Sears, Chris White und ihre Kollegen sehr viel Erfahrung in Google ein. Rubin beispielsweise hatte schon in den 1990ern an Magic Cap, einem Betriebssystem für mobile Geräte mit grafischer Benutzeroberfläche, gearbeitet. Auch seine Firma Danger Inc. produzierte mobile Geräte. Das erstmals 2002 erschienene Hiptop war ein Telefon mit Organizer-Fähigkeiten.

Nach dem Kauf von Android Inc. arbeitete man bei Google in aller Stille weiter – woran, ist seit der Pressemitteilung der OHA bekannt.

1.1.3 Evolution einer Plattform

Zwischen der Ende 2007 veröffentlichten Vorschau des Android SDK und der im G1 eingesetzten Version lagen noch einmal viele Monate Entwicklungsarbeit. Google war bewusst frühzeitig auf interessierte Programmierer zugegangen, um deren Rückmeldungen in die Plattform einarbeiten zu können und natürlich um Appetit auf Android zu machen. Zudem hatte man 2008 zum ersten Mal einen Entwickler-Wettbewerb gestartet und ein hohes Preisgeld ausgelobt. Der Suchmaschinenriese nutzte den Kontakt zu den Finalisten, um weitere Fehler zu beheben und um die Programmierschnittstellen zu verfeinern.

Ein direkter Vergleich der ersten Android-Version mit dem damals verfügbaren iPhone-Betriebssystem fiel natürlich zugunsten des Apple-Produkts aus. Dies ist nicht verwunderlich, hatte der kalifornische Computerbauer sein edles Smartphone doch schon viele Monate vorher veröffentlicht und demzufolge ausreichend Zeit für die Detailpflege gehabt.

Mittlerweile liegen die beiden Kontrahenten gleich auf. Google hat mit beachtlichem Tempo an der Plattform gearbeitet und dabei nicht nur Fehler beseitigt, sondern auch die mitgelieferten Anwendungen konsequent verbessert sowie unzählige neue Funktionen eingeführt. Oftmals lagen zwischen zwei Releases, die traditionell den Namen einer Süßspeise tragen, nur wenige Monate.

Cupcake (Android 1.5) führte unter anderem das sogenannte *Input Method Framework* ein und ermöglichte damit erstmals Geräte ohne Hardwaretastatur. Ferner fand schon zu diesem Zeitpunkt eine (allerdings kaum genutzte) Spracherkennungsfunktion ihren Weg in die Plattform.

Donut (Android 1.6) unterstützte Geräte mit unterschiedlichen Anzeigegrößen und Auflösungen. Ferner erweiterte Google die Plattform um eine Sprachsynthesesoftware für Deutsch, Italienisch, Englisch, Französisch und Spanisch. Außerdem hatten Entwickler erstmals Zugriff auf eine Gestenerkennung und konnten die systemweite Schnellsuche um Inhalte ihrer Anwendungen erweitern.

Unter dem Namen *Eclair* werden die Versionen 2.0, 2.0.1 und 2.1 zusammengefasst. Mit Android 2.0 hielt eine vollständig neue Programmierschnittstelle für den Zugriff auf Kontakt- und Kontendaten Einzug. Gleichzeitig wurden die bisherigen Klassen für veraltet erklärt. Für den Anwender bot dies den großen Vorteil, sein Mobiltelefon auch mit anderen Diensten, beispielsweise Facebook, synchronisieren zu können.

Allerdings machen die noch immer unzureichend dokumentierten neuen Klassen vielen Programmierern das Leben unnötig schwer. Kapitel 16, »Kontakte und Organizer«, zeigt Ihnen den souveränen Umgang mit Kontakten. Android 2.1 führte die bei Endanwendern sehr beliebten Live Wallpaper ein. Wie Sie solche animierten Hintergründe selbst erstellen, zeige ich Ihnen in Kapitel 8, »Widgets und Wallpapers«.

Mit *Froyo* (Version 2.2) bekam Android eine zentrale Schnittstelle für das Sichern und Wiederherstellen von Anwendungsdaten. Außerdem können Programme nun auch auf Wechselmedien installiert werden. Was Sie hierbei beachten müssen, beschreibe ich ausführlich in Kapitel 10, »Das Android-Dateisystem«.

Gingerbread (Android 2.3.x) enthält einen vollständigen SIP (Session Initiation Protocol)-Stapel und bindet VoIP-gestützte Internettelefonie in die Plattform ein. Neben einem Download-Manager, den alle Apps nutzen können, wurde die Kameraunterstützung erweitert und die Geschwindigkeit des Systems weiter verbessert.

Honeycomb (Versionen 3.0, 3.1 und 3.2) wurde speziell für Tablets optimiert. Google hat die Plattform hierzu um zahlreiche Konzepte erweitert. Beispielsweise ermöglichen *Fragmente* die Wiederverwendung von Teilfunktionen einer Activity. Die *Action Bar* löst nicht nur die bislang bekannten klassischen Menüs ab, sondern führt eine einheitliche Navigation innerhalb von Anwendungen ein. Außerdem entfällt mit Honeycomb die Notwendigkeit der klassischen Hardware-Buttons. Die *System Bar* kombiniert virtuelle Knöpfe mit Benachrichtigungen.

Ice Cream Sandwich (Version 4.0.x) führt die seit Version 3.x existierenden zwei Entwicklungslinien (Smartphones und Tablets) wieder zusammen. Damit profitieren alle Geräteklassen von den kurz angesprochenen neuen Funktionen. Die Plattform sowie die mitgelieferten Anwendungen wurden weiter poliert und noch bedienfreundlicher gestaltet. Außerdem ist endlich der Zugriff auf Kalendereinträge möglich.

Kurz vor Drucklegung, auf Googles Entwicklerkonferenz I/O 2012, hat der Suchmaschinenprimus *Jelly Bean* (Android 4.1) vorgestellt. Das Hauptaugenmerk dieses Releases galt der Steigerung der Geschwindigkeit; vor allem Animationen wirken geschmeidiger. Außerdem reagiert das System noch schneller auf Gesten. Eine neue Sprachsuche macht Apples Siri Konkurrenz. Und Google Now möchte dem Nutzer situationsbezogene Informationen bieten.

Der häufige Release-Wechsel hat aus Android in beeindruckend kurzer Zeit eine stabile und anwenderfreundliche Plattform für mobile Geräte gemacht. Leider dauert es

nach wie vor oft viele Monate, bis Hersteller von Smartphones und Tablets ihren Kunden entsprechende Aktualisierungen zur Verfügung stellen. Nicht wenige gehen leer aus. Der Hauptgrund hierfür liegt in der bereits angesprochenen Freiheit, die Plattform nahezu nach Belieben verändern zu können. Zahlreiche Gerätebauer haben nämlich nicht nur eigene Apps, Widgets und Wallpaper hinzugefügt, sondern auch bestehende Programme (zum Teil erheblich) verändert. Dies betrifft nicht nur die mitgelieferten Standardanwendungen, sondern auch die Android-Benutzeroberfläche. Hersteller müssen ihre speziellen Änderungen aber in jedes neue Android-Release, das Google veröffentlicht, aufs Neue einarbeiten. Es liegt auf der Hand, dass dieser Prozess aufwendig und zeitraubend ist.

Für Sie als Entwickler sind solche Erweiterungen in der Regel transparent, zumindest solange die Gerätebauer die Programmierschnittstellen unangetastet lassen. Allerdings kommen neue Android-Versionen mit deutlicher Verzögerung beim Endanwender an. Aus diesem Grund ist es ratsam, sich regelmäßig auf *http://developer.android.com/resources/dashboard/platform-versions.html* über den aktuellen Verbreitungsgrad der verschiedenen Android-Versionen zu informieren. Denn erst ab einer bestimmten Anzahl von potenziellen Nutzern lohnt der Einsatz neuer Programmierschnittstellen oder Funktionen.

1.2 Systemarchitektur

Vielleicht fragen Sie sich, was das Wort Plattform im Zusammenhang mit Android bedeutet. Handelt es sich nicht einfach um ein Betriebssystem für mobile Geräte, für das Sie Programme in Java schreiben?

1.2.1 Überblick

Aus der Sicht des Endanwenders bildet Android eine mittlerweile sehr große Klasse von mobilen Geräten, beispielsweise Smartphones, Netbooks und Tablets. Zahlreiche Hersteller bieten Modelle in unterschiedlichsten Ausstattungsvarianten an. Neben preisgünstigen Einsteigerprodukten finden sich im Hochpreis-Segment Geräte mit viel Arbeitsspeicher, großen, auflösungsstarken Bildschirmen und hoher Prozessorleistung. Auf allen Produkten läuft Android.

Dennoch ist Android nicht nur ein Betriebssystem. Zu Android gehören nämlich eine Reihe von Standardanwendungen, beispielsweise ein Anwendungsstarter mit Unterstützung für Widgets, eine Kontaktdatenbank, eine Uhr mit Weckfunktion, ein Browser und ein E-Mail-Client. Sehr oft ist auch *Play Store* bzw. *Google Play* enthalten, eine Anwendung zum Kaufen und Herunterladen von Programmen und anderen

Medien. Die Plattform *Google Play* fasst Dienste wie *Android Market* und *Google Music* unter einer gemeinsamen Oberfläche zusammen.

Ebenfalls Bestandteil von Android, allerdings für den Endanwender nicht sichtbar, ist die virtuelle Maschine Dalvik. Sie führt nahezu[4] alle Programme aus, die der Benutzer auf einem Android-System startet. Dies betrifft die weiter oben genannten Standardanwendungen, aber auch selbst geschriebene Programme. Ihre Software wird also nicht unmittelbar durch den Prozessor des Mobiltelefons abgearbeitet, sondern von einer Art Zwischenschicht interpretiert. Java-Entwickler kennen dies. Quelltext wird durch einen Compiler in Bytecode umgewandelt, den die *Java Virtual Machine (JVM)* später ausführt. Aus der Sicht des Betriebssystems ist diese ein gewöhnliches Programm.

Android funktioniert auf sehr ähnliche Weise. Sie entwickeln Ihre Programme in der Sprache Java und wandeln die Quelltexte in sogenannte Dalvik Executables um. Diese werden durch die virtuelle Maschine Dalvik abgearbeitet. Seit Android 2.2 steht ihr ein Just-in-time-Compiler zur Verfügung. Er wandelt, vereinfacht ausgedrückt, den Bytecode zur Laufzeit in ein noch schneller ausführbares Format um.

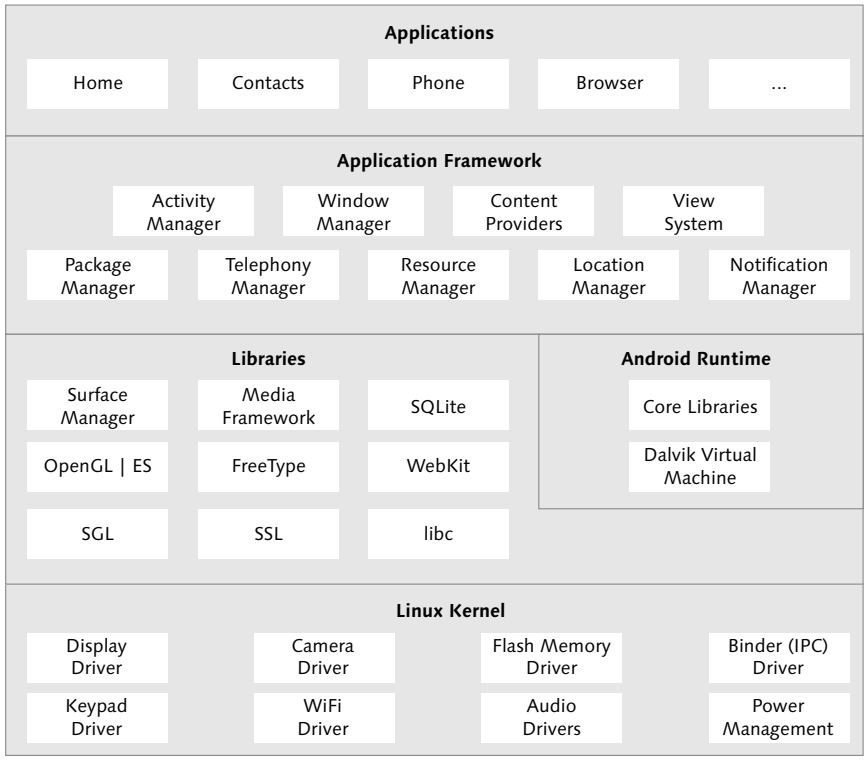


Abbildung 1.1 Schematischer Aufbau der Android-Plattform

4 Eine Ausnahme bilden (Teile von) Apps, die mit dem Android Native Development Kit geschrieben wurden.

Wenn, wie weiter oben beschrieben, die virtuelle Maschine ein mehr oder weniger normales Programm ist, benötigt sie ein Betriebssystem, das Systemdienste wie Sicherheit, Speicher- und Prozessverwaltung, Treibermodell und Netzwerkstapel zur Verfügung stellt. Android nutzt hierfür Linux 2.6. Das Open-Source-Betriebssystem fungiert aber auch als Abstraktionsschicht, die die Hardware vor den übrigen Teilen der Plattform kapselt. Auf diesem Fundament setzen weitere Schichten auf, die in Abbildung 1.1 zu sehen sind.

Android enthält eine Reihe von C/C++-Bibliotheken, die von verschiedenen Komponenten der Plattform genutzt werden. Entwickler greifen indirekt über das Android-Anwendungsframework auf sie zu:

- Die SystemC Library ist eine Implementierung der Standard C-Bibliothek, die speziell an mobile Geräte angepasst wurde.
- Die Media Libraries basieren auf PacketVideos OpenCORE-Bibliotheken. Sie ermöglichen die Aufnahme und Wiedergabe von zahlreichen populären Audio-, Video- und Grafikformaten (unter anderem MPEG4, H.264, MP3, AAC, AMR, JPG und PNG).
- Der Surface Manager koordiniert Bildschirmzugriffe und fügt die 2D- und 3D-Ausgaben verschiedener Anwendungen zu einem Gesamtbild zusammen.
- LibWebCore ist eine Rendering Engine für Webinhalte.
- SGL ist eine 2D-Grafikbibliothek.
- Bei den 3D libraries handelt es sich um eine auf OpenGL ES basierende 3D-Grafikbibliothek. Sofern Hardwarebeschleunigung zur Verfügung steht, wird diese automatisch genutzt.
- FreeType ist eine Rendering Engine für Bitmap- und Vektorzeichensätze.
- SQLite ist eine leichtgewichtige relationale Datenbank.

Die übrigen Schichten der Android-Plattform stelle ich Ihnen in den folgenden Abschnitten vor.

1.2.2 Android Runtime

Wie Sie bereits wissen, werden Android-Programme durch die virtuelle Maschine Dalvik ausgeführt. Sie wurde ursprünglich von Dan Bornstein entwickelt und nach einem kleinen Fischerort auf Island benannt, in dem Vorfahren seiner Familie lebten. Da Dalvik ausdrücklich für den Einsatz in mobilen Geräten konzipiert wurde, unterscheidet sie sich in einigen Punkten wesentlich von klassischen Java-Laufzeitumgebungen.

Dalvik ist registerbasiert, hat ein eigenes Bytecode-Format sowie einen eigenen Befehlssatz. *.class*-Dateien werden deshalb mit einem Tool namens *dx* in sogenannte

Dalvik Executables (.dex) umgewandelt. Als Entwickler müssen Sie sich darum aber nicht kümmern. Dies erledigen die Werkzeuge, die ich Ihnen in Abschnitt 1.3, »Entwicklungswerkzeuge«, ausführlicher vorstellen werde, automatisch. *.dex*-Dateien sind sehr kompakt. Beispielsweise können mehrere Klassen in einer solchen zusammengefasst werden. Sich wiederholende Zeichenketten und andere Konstanten erscheinen nur einmal.

Dalvik wurde aber nicht nur auf einen minimalen Speicherverbrauch hin optimiert. Bei der Entwicklung spielte auch eine effiziente Ausführbarkeit von mehreren virtuellen Maschinen eine große Rolle. Denn jede Android-Anwendung läuft als eigener Prozess mit jeweils eigener Instanz der Dalvik Virtual Machine. Dies sorgt für Sicherheit und erhöht die Stabilität der Plattform.

Den zweiten Baustein der Android Runtime bilden die sogenannten *Core Libraries*. Sie enthalten unter anderem (die folgende Aufzählung ist bewusst nicht vollständig) die dem Java-Entwickler vertrauten Pakete `java.lang`, `java.io`, `java.math`, `java.net`, `java.security`, `java.util` und `java.text`.

Android verwendet hier eine Untermenge der Klassenbibliothek der (seit Ende 2011 nicht mehr weiterentwickelten) freien Java-Implementierung Apache Harmony. Für die Realisierung der Benutzeroberfläche, die Steuerung des Lebenszyklus einer Anwendung, Telefoniefunktionen sowie Multimedia werden aber Android-eigene Programmierschnittstellen angeboten. Das hierfür zuständige Application Framework stelle ich Ihnen im folgenden Abschnitt vor.

1.2.3 Application Framework

Mit Hilfe des *Application Frameworks* lassen sich äußerst komfortable, ästhetische und leicht bedienbare mobile Anwendungen mit großem Funktionsumfang erstellen. Sie haben Zugriff auf die Gerätehardware, zum Beispiel Kamera, Netzwerk oder Sensoren. Auch das Lesen und Schreiben von Kontaktdaten oder Terminen ist bequem möglich. Ein ausgefeiltes, leicht handhabbares Rechtesystem steuert hierbei, was ein Programm tun darf.

Eines der Kernkonzepte des Application Frameworks ist, dass Anwendungen ihre Funktionen veröffentlichen, also anderen Programmen verfügbar machen können. Da Anwendungen von Drittanbietern den Android-Standardanwendungen gleichgestellt sind, kann der Benutzer sehr leicht den Webbrowser, den E-Mail-Client oder den Mediaplayer austauschen.

Aber auch das Ersetzen von einzelnen Programmfunktionen (beispielsweise das Verfassen einer SMS) ist möglich. Selbstverständlich können Programme umgekehrt auch Funktionen anderer Anwendungen anfordern. Auch dies wird über das bereits erwähnte Rechtesystem gesteuert.

Kernbestandteile des Application Frameworks sind unter anderem:

- *Views* bilden die Basis für alle Benutzeroberflächen. Android bietet zahlreiche Standardbedienelemente, wie etwa Textfelder, Schaltflächen, Ankreuzfelder und Listen. Bestehende Views können durch sogenannte Themes nahezu beliebig angepasst werden. Natürlich sind vollständig eigenentwickelte Views realisierbar.
- *Content Provider* gestatten Anwendungen den Zugriff auf Daten anderer Programme. Auch das Bereitstellen der eigenen Anwendungsdaten ist auf diese Weise leicht möglich.
- Der *Resource Manager* gewährt Zugriff auf lokalisierte Zeichenketten, auf Grafiken und Layoutdateien.
- Der *Notification Manager* bietet Anwendungen den Zugriff auf die Android-Statuszeile. Mit ihm können auch kleine Popup-Nachrichten erzeugt werden.
- Der *Activity Manager* steuert den Lebenszyklus einer Anwendung.

Alle hier aufgeführten Bestandteile werden in den folgenden Kapiteln ausführlich vorgestellt. Zunächst möchte ich Sie aber mit den Entwicklungswerkzeugen bekannt machen. Wie Sie bald sehen werden, sorgen diese für eine komfortable und effiziente Programmierarbeit.

1.3 Entwicklungswerkzeuge

Das *Android Software Development Kit (SDK)* bildet die Grundlage für die Anwendungsentwicklung. Es ist in Versionen für Windows, Linux und Mac OS X erhältlich. Um die Software nutzen zu können, muss auf Ihrem Rechner eine Ausgabe des *Java Development Kits (JDK)* 5 oder 6 installiert sein. Ich empfehle JDK 6.

Eine Java-Laufzeitumgebung (JRE) reicht nicht aus, da sie beispielsweise den Java-Compiler *javac* nicht enthält. Windows- und Linux-Nutzer können das SDK von *www.oracle.com* herunterladen. Die zum Zeitpunkt der Drucklegung aktuelle Fassung finden Sie auf der Begleit-DVD des Buches im Verzeichnis *Software*. Falls Sie unter Mac OS X entwickeln, besuchen Sie *developer.apple.com*.

Ferner benötigen Sie die Entwicklungsumgebung Eclipse. Theoretisch könnten Sie Ihre Quelltexte auch mit einem beliebigen anderen Editor schreiben und »nur« das Android SDK verwenden. Allerdings bietet Google mit den *Android Development Tools (ADT)* ein äußerst mächtiges Plug-in für Eclipse an, das das Schreiben, Debuggen und Veröffentlichen von Apps stark vereinfacht. Ich beziehe mich deshalb im weiteren Verlauf des Buches ausschließlich auf das Gespann Eclipse und ADT. Sie finden Eclipse 3.7 in Versionen für Windows, Linux und Mac OS X im Verzeichnis *Software* der Begleit-DVD.

Installieren Sie nun Java und Eclipse, falls die Software nicht in geeigneten Versionen auf Ihrem Entwicklungsrechner vorhanden ist. Achten Sie darauf, dass der Unterordner *bin* des JDK-Installationsverzeichnisses im Standardsuchpfad enthalten ist. Gegebenenfalls erweitern Sie die *Umgebungsvariable* PATH um einen entsprechenden Verweis.

Tipp

Zum Java Development Kit gehört eine sehr umfangreiche Dokumentation. Ich rate Ihnen, das entsprechende Archiv herunterzuladen und im Installationsverzeichnis des JDK zu entpacken.

1.3.1 Android SDK

Das Android SDK besteht aus einem Emulator, zahlreichen Tools, sogenannten Plattformen sowie Dokumentation und Beispielen. Plattformen entsprechen den bekannten Android-Versionen, also Cupcake (1.5), Froyo (2.2) oder Ice Cream Sandwich (4.x). Entpacken Sie nun das für Ihr Betriebssystem geeignete Archiv (Sie finden Versionen für Windows, Linux und Mac OS X auf *developer.android.com* sowie im Verzeichnis *Software* der Begleit-DVD) an einer beliebigen Stelle.

Unter Windows bietet sich *C:\Programme* an. Linux- und Mac OS X-Anwender können es unter */opt* ablegen. Beachten Sie, dass Sie für den schreibenden Zugriff auf diese Verzeichnisse Administratorrechte benötigen. Abbildung 1.2 zeigt die entstandene Ordnerstruktur.

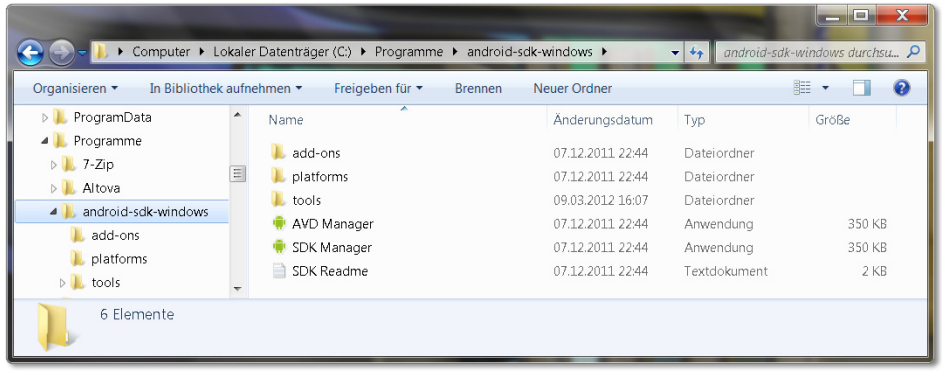


Abbildung 1.2 Verzeichnisstruktur des Android SDK Starter Package

Da mit jeder neuen Version die Größe des SDKs zunahm, hat sich Google entschlossen, zum Download nur noch ein sogenanntes Starter Package anzubieten. Es enthält keine Plattformen. Das korrespondierende Verzeichnis *platforms* ist also leer. Diese werden mit dem *SDK Manager* nachgeladen.

Bevor Sie dieses Programm starten, sollten Sie die Ordner *tools* und *platform-tools* in den Standardsuchpfad aufnehmen, indem Sie die Umgebungsvariable PATH entsprechend erweitern. Dies ist nützlich, wenn Sie Programme des Android SDKs von der Kommandozeile aus aufrufen oder in Shellscripts einsetzen möchten. Bitte beachten Sie, dass *platform-tools* erst nach der im Folgenden beschriebenen Installation vorhanden ist.

Starten Sie nun den SDK Manager. Er prüft, ob neue oder aktualisierte Komponenten zum Download zur Verfügung stehen. Das in Abbildung 1.3 gezeigte Programm zeigt die zum Zeitpunkt der Drucklegung gefundenen Pakete an.

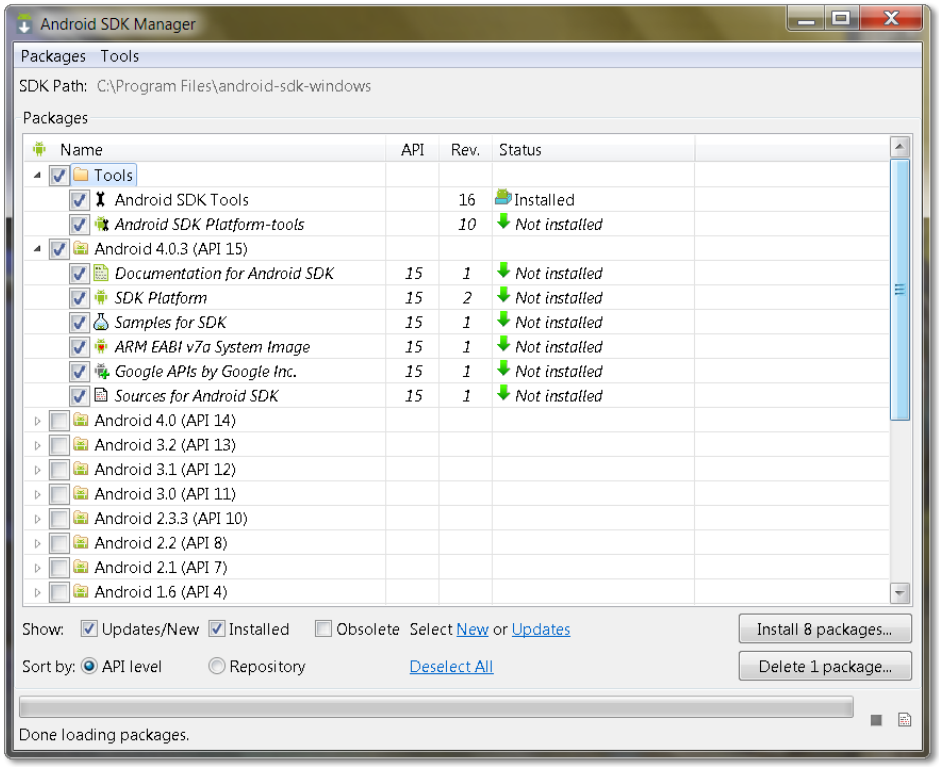


Abbildung 1.3 Auswahl der zu installierenden Pakete

Um ein Paket herunterzuladen oder zu aktualisieren, setzen Sie ein Häkchen vor dem entsprechenden Eintrag. Mit DESELECT ALL können Sie alle Pakete auf einmal deselektieren. Das ist praktisch, wenn Sie nur ein bestimmtes Paket installieren möchten. Mit NEW kennzeichnen Sie alle neuen Pakete für den Download. INSTALL PACKAGES startet den Vorgang.

Neben den zahlreichen Plattformen steht unter anderem ein USB-Treiber für Windows, eine Sammlung von Beispielen und natürlich die ausführliche Dokumentation

zum Herunterladen bereit. Eine schnelle Internetverbindung und reichlich Plattenplatz vorausgesetzt, rate ich Ihnen aus Gründen der Bequemlichkeit dazu, schon jetzt alle angebotenen Komponenten zu installieren. Spätestens wenn Sie eine Anwendung für Google Play vorbereiten, müssen Sie Ihr Werk nämlich unter möglichst vielen Versionen testen.

Sollten Sie ältere Plattformen zunächst nicht herunterladen wollen, können Sie dies später problemlos nachholen. Die Android SDK Tools, Android SDK Platform Tools, die Plattform-Version 4.0.3 einschließlich Dokumentation und Quelltexte (Sources for Android SDK) sowie die Beispiele sollten Sie aber auf jeden Fall installieren.

Nachdem alle Pakete heruntergeladen und installiert wurden, beenden Sie den SDK Manager und starten das in Abbildung 1.4 gezeigte Programm *AVD Manager*. Sie finden es ebenfalls im Installationsverzeichnis des Android SDKs.

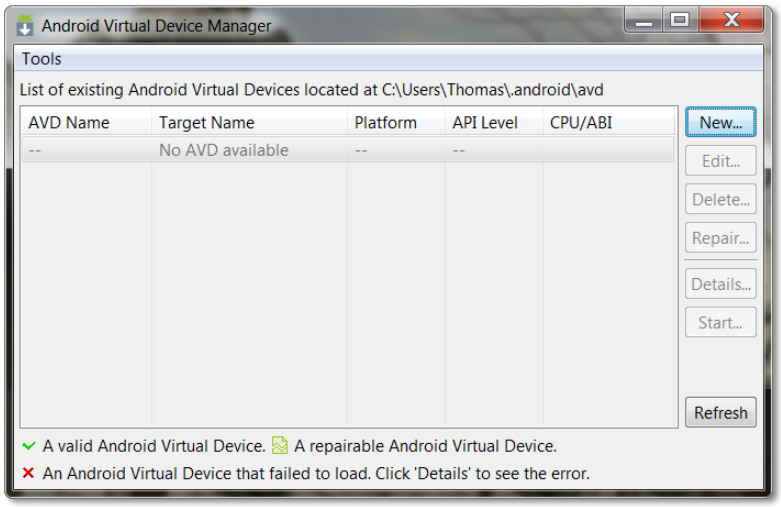


Abbildung 1.4 Der Android Virtual Device Manager

Android Virtual Devices (AVD) fassen bestimmte Einstellungen des im SDK enthaltenen Emulators zusammen, um ein bestimmtes physikalisches Gerät nachzubilden. Dazu gehören unter anderem:

- ein Hardwareprofil: Hat das simulierte Gerät beispielsweise eine Kamera und eine physikalische Tastatur?
- das zu verwendende Systemabbild, also die zu simulierende Android-Version
- die zu verwendende Emulator-Skin, vereinfacht ausgedrückt eine Grafik, die um den simulierten Gerätebildschirm gezeichnet wird, um den Eindruck zu erwecken, man nutze ein reales Gerät

Um ein AVD anzulegen, klicken Sie auf NEW. Übernehmen Sie die Angaben aus Abbildung 1.5, und schließen Sie Ihre Eingabe mit CREATE AVD ab. Das auf diese Weise erzeugte virtuelle Gerät erscheint im Hauptfenster des Android Virtual Device Managers unter LIST OF EXISTING ANDROID VIRTUAL DEVICES LOCATED AT.... Mit Doppelklick öffnen Sie einen kleinen Dialog mit Informationen zu einem AVD.

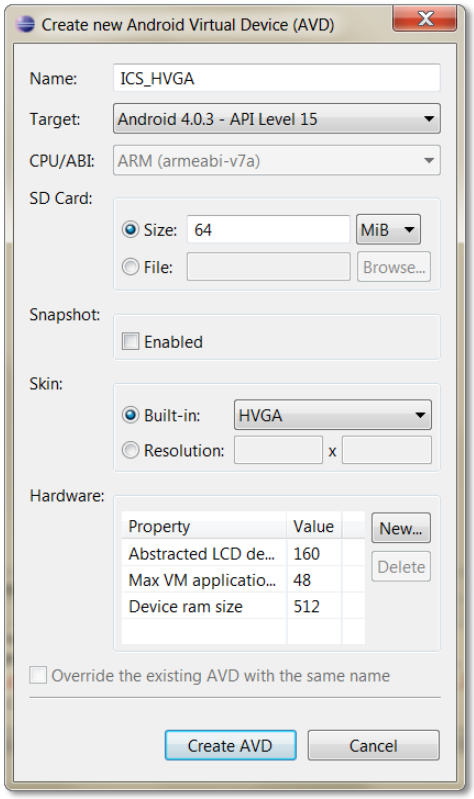


Abbildung 1.5 Dialog zum Anlegen eines virtuellen Geräts

Um das virtuelle Gerät zu starten, öffnen Sie die *Eingabeaufforderung* bzw. eine *Shell* und geben `emulator -avd ICS_HVGA` ein. Sie können den Emulator beenden, indem Sie einfach das Konsolenfenster schließen. Falls das Starten des Emulators nicht klappt, prüfen Sie, ob Sie die *Umgebungsvariable* `PATH` wie beschrieben um die Verzeichnisse *bin* des JDKs sowie *tools* des SDKs erweitert haben.

Sie haben in diesem Abschnitt das Android SDK installiert und den Emulator mit der Android-Version 4.0.3 Probe gefahren. Als Letztes müssen Sie nur noch die Eclipse-Plugin-Sammlung *Android Development Tools (ADT)* einrichten.

1.3.2 Android Development Tools

Starten Sie Eclipse, und öffnen Sie mit HELP • INSTALL NEW SOFTWARE den in Abbildung 1.6 gezeigten Dialog INSTALL.

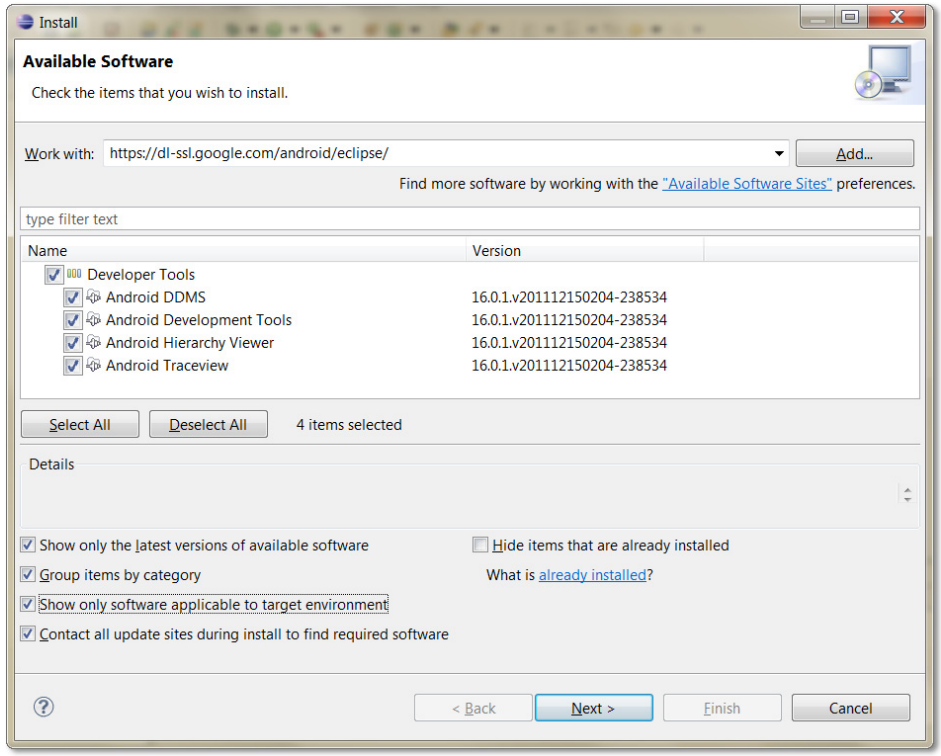


Abbildung 1.6 Installation der Android Development Tools

Geben Sie in das Eingabefeld WORK WITH die Adresse *https://dl-ssl.google.com/android/eclipse/* ein, und drücken Sie anschließend ↵. Sie haben auf diese Weise eine *Remote Update Site* hinzugefügt.

Setzen Sie nun ein Häkchen vor DEVELOPER TOOLS, und klicken Sie anschließend auf NEXT. Nun folgen Sie den weiteren Anweisungen des Installationsassistenten. Nach einem Neustart der IDE finden Sie im Einstellungsdialog, den Sie mit WINDOW • PREFERENCES öffnen, den Knoten ANDROID.

Dort tragen Sie unter SDK LOCATION den Pfad ein, in dem Sie das Android SDK entpackt hatten, zum Beispiel *C:\Programme\android-sdk-windows*. Falls im Anschluss daran die Liste der SDK TARGETS leer ist, klicken Sie auf APPLY. Wählen Sie nun ANDROID 4.0.3 durch Anklicken als Standardplattform aus, und schließen Sie danach den Dialog mit OK.

1.3.3 Das erste eigene Projekt

Öffnen Sie mit FILE • NEW • PROJECT den Assistenten zum Anlegen von Projekten, und markieren Sie im Knoten ANDROID den Eintrag ANDROID PROJECT. NEXT öffnet die zweite Seite des Dialogs, die Sie in Abbildung 1.7 sehen. Klicken Sie nun auf CREATE PROJECT FROM EXISTING SAMPLE und danach erneut auf NEXT. Auf der Seite *Select Build Target* setzen Sie ein Häkchen vor ANDROID 4.0.3. Nach einem weiteren NEXT müssen Sie auf der Seite *Select Sample* nur noch den Eintrag LUNARLANDER auswählen und den Dialog mit FINISH abschließen.

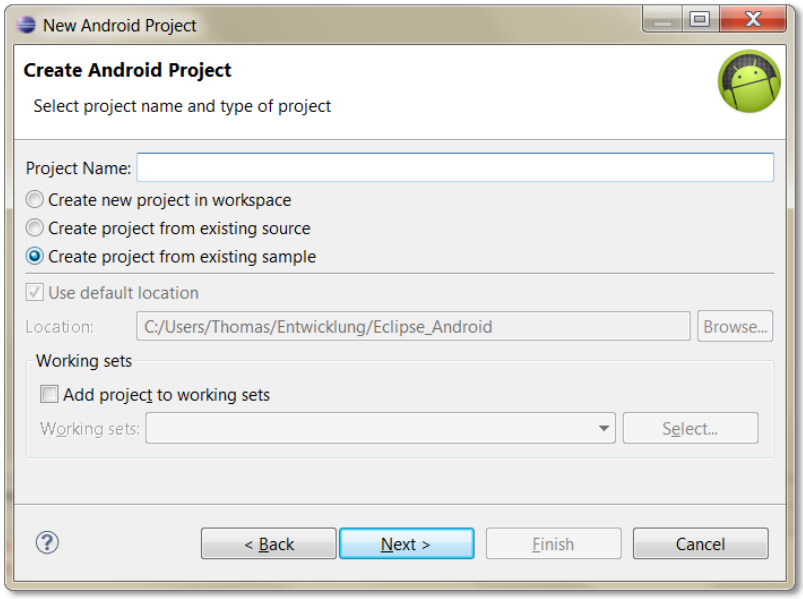


Abbildung 1.7 Assistent zum Anlegen eines neuen Projekts

Nachdem das Projekt angelegt wurde, ist es im sogenannten *Package Explorer* zu sehen. Falls diese Sicht nicht angezeigt wird, können Sie sie mit WINDOW • SHOW VIEW • PACKAGE EXPLORER öffnen. Der Package Explorer stellt alle relevanten Dateien und Verzeichnisse von Projekten in einer baumartigen Struktur dar und wird zu Ihrem Navigationszentrum während der Arbeit an einer App.

Um *LunarLander* im Emulator zu testen, klicken Sie im Package Explorer mit der rechten Maustaste auf den Eintrag LUNARLANDER und wählen dann RUN AS • ANDROID APPLICATION. Da Sie Android 4.0.3 als Standard-Target ausgewählt haben, sollte sich automatisch der in Abbildung 1.8 gezeigte Emulator mit dem weiter vorne erzeugten AVD öffnen.

Abbildung 1.8 LunarLander im Emulator

1.3.4 Android-Quelltexte einhängen

Nicht nur bei der Fehlersuche ist es manchmal wünschenswert, in Systemklassen hineinsehen zu können. Das Studium der zur Plattform gehörenden Klassen kann gerade Android-Einsteigern wertvolle Tipps für die Programmierung liefern. Erstaunlicherweise war das lange Zeit mit vergleichsweise hohem Aufwand verbunden. Mittlerweile genügt es aber, im *Android SDK Manager* für die gewünschte Plattform das Paket *Sources for Android SDK* herunterzuladen. Im Verzeichnis *sources* des Installationsverzeichnisses befindet sich danach ein entsprechender Unterordner, zum Beispiel *android-15* für Android 4.0.3. Dessen Pfad muss in Eclipse dem Archiv *android.jar* als sogenanntes *Source attachment* zugewiesen werden.

Wählen Sie nach einem Rechtsklick auf die in der Sicht *Package Explorer* angezeigte Projektwurzel (*LunarLander*) den Menüpunkt Build Path • Configure Build Path aus und tragen anschließend auf der Registerkarte Libraries des Dialogs *Properties for LunarLander* den Pfad ein. Öffnen Sie hierzu die Knoten für die gewünschte Plattform (zum Beispiel Android 4.0.3) sowie *android.jar*. Klicken Sie nun auf Source attachment, anschließend auf Edit.

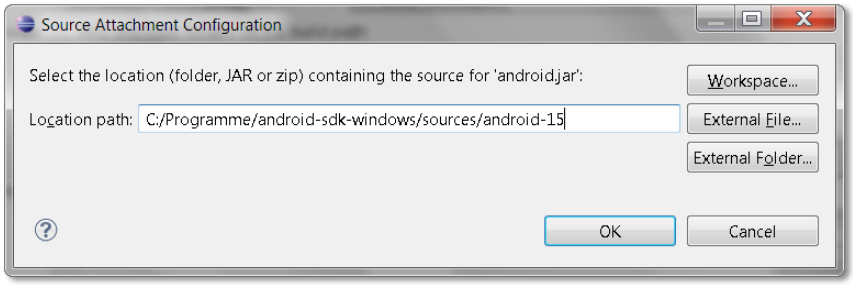


Abbildung 1.9 Der Dialog Source Attachment Configuration

Jetzt können Sie in dem in Abbildung 1.9 gezeigten Dialog *Source Attachment Configuration* den Pfad (auf meinem System ist dies für Android 4.0.3 *C:/Programme/android-sdk-windows/sources/android-15*) eingeben.

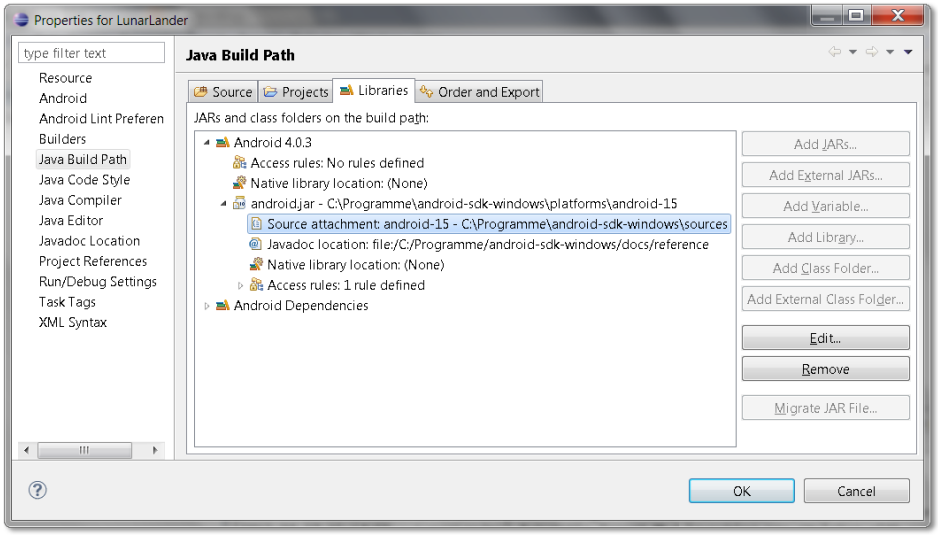


Abbildung 1.10 Der Dialog Properties for LunarLander

Die Änderungen erscheinen im Dialog *Properties for LunarLander*, der in Abbildung 1.10 zu sehen ist. Nachdem Sie auch diesen mit OK geschlossen haben, können Sie einen Blick auf eine Systemklasse werfen. Öffnen Sie hierzu mit NAVIGATE • OPEN TYPE den in Abbildung 1.11 gezeigten Dialog und geben unter *Enter name prefix or pattern* `android.app.Activity` ein.

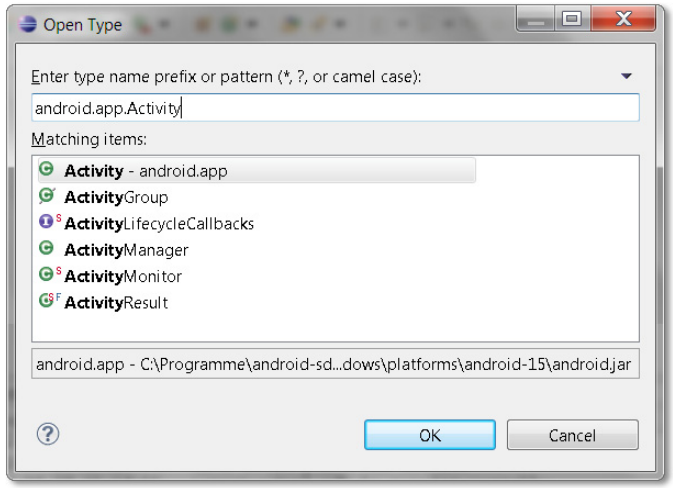


Abbildung 1.11 Der Dialog Open Type

OK schließt den Dialog und zeigt die ausgewählte Klasse in einem schreibgeschützten Editorfenster an. Wie dies aussehen kann, ist in Abbildung 1.12 zu sehen.

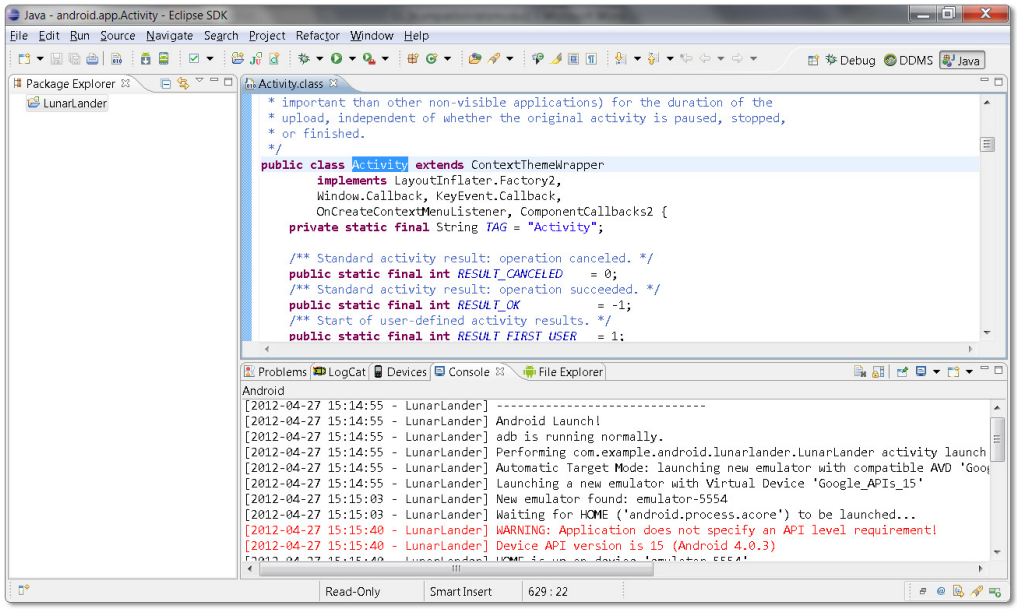


Abbildung 1.12 Editorfenster mit dem Quelltext der Systemklasse Activity

Kapitel 2
Hallo Android!

Die erste eigene App ist schneller fertig, als Sie vielleicht glauben. Dieses Kapitel führt Sie in leicht nachvollziehbaren Schritten zum Ziel.

Seit vielen Jahrzenten ist es schöne Tradition, anhand des Beispiels »Hello World!« in eine neue Programmiersprache oder Technologie einzuführen. Dahinter steht die Idee, erste Konzepte und Vorgehensweisen in einem kleinen, überschaubaren Rahmen zu demonstrieren.

Android bleibt dieser Tradition treu. Wenn Sie mit dem Eclipse-Projektassistenten ein neues Projekt anlegen, erzeugen die Android Development Tools eine minimale, aber lauffähige Anwendung. Sie gibt einen Text aus, der mit den Worten »Hello World« beginnt. Im Verlauf dieses Kapitels erweitern Sie diese Anwendung um die Möglichkeit, einen Nutzer namentlich zu begrüßen. Ein Klick auf FERTIG schließt die App.

Hinweis

Sie finden die vollständige Version des Projekts *Hallo Android* im Verzeichnis *Quelltexte* der Begleit-DVD. Um mit den Entwicklungswerkzeugen vertraut zu werden, rate ich Ihnen aber, sich diese Fassung erst nach der Lektüre dieses Kapitels in Ihren Eclipse-Arbeitsbereich zu kopieren.

2.1 Android-Projekte

Projekte fassen alle Artefakte einer Android-Anwendung zusammen. Dazu gehören unter anderem Quelltexte, Konfigurationsdateien, Testfälle, aber auch Grafiken, Sounds und Animationen. Natürlich sind Projekte keine Erfindung der Android Development Tools (ADT), sondern bilden eines der Kernkonzepte von Eclipse.

Sie werden im *Arbeitsbereich* (engl. *workspace*) abgelegt. Grundsätzlich können Sie mit beliebig vielen Projekten gleichzeitig arbeiten. Für das Öffnen und Schließen von Projekten ist der Package Manager zuständig.

2.1.1 Projekte anlegen

Um ein neues Projekt anzulegen, wählen Sie in Eclipse FILE • NEW • PROJECT. Klicken Sie auf ANDROID • ANDROID PROJECT, und wechseln Sie mit NEXT auf die zweite Seite des Assistenten. Dort markieren Sie CREATE NEW PROJECT IN WORKSPACE und vergeben einen Projektnamen. Ich habe zum Beispiel *Hallo Android* gewählt. Der neben LOCATION angezeigte Pfad verweist auf das Projektwurzelverzeichnis. Sofern Sie das Häkchen vor USE DEFAULT LOCATION nicht entfernt haben (was Sie auch nicht tun sollten), ist dies ein Unterverzeichnis des Eclipse-Arbeitsbereichs. Dieser liegt standardmäßig im Ordner *workspace* unterhalb Ihres Heimatverzeichnisses. Wenn Sie mit mehreren solcher Workspaces parallel arbeiten, können Sie (vor dem Aufruf des Projektassistenten) mit FILE • SWITCH WORKSPACE • OTHER einen anderen Pfad einstellen. Fürs Erste sollten Sie diese Einstellungen aber unverändert lassen.

Mit NEXT gelangen Sie zur Seite *Select Build Target*, die Sie schon aus dem ersten Kapitel kennen. Durch die Auswahl einer bestimmten Plattform legen Sie die Android-Version fest, gegen die Sie entwickeln. Welche Einträge hier erscheinen, hängt davon ab, wie viele Plattformen Sie mit Hilfe des SDK Managers herunter geladen haben. Setzen Sie ein Häkchen vor ANDROID 4.0.3 und klicken abermals auf NEXT. Auf der in Abbildung 2.1 gezeigten Seite *Application Info* machen Sie wichtige Angaben zur neuen App.

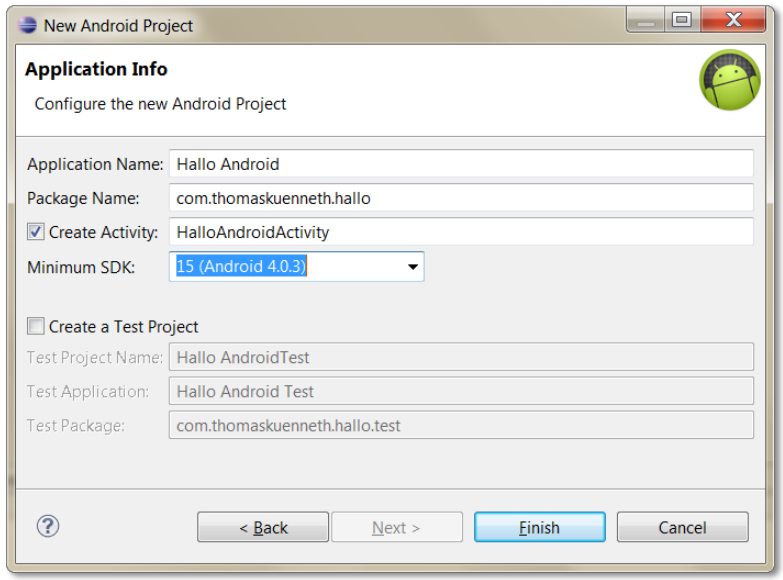


Abbildung 2.1 Anlegen des Projekts »Hallo Android«

Der Application name wird später auf dem Gerät bzw. im Emulator angezeigt. Bei der Vergabe des Package name sollten Sie besonders sorgfältig vorgehen, vor allem wenn Sie eine Anwendung in Google Play veröffentlichen möchten. Denn der hier eingetragene Paketname referenziert genau eine App, muss also eindeutig sein. Idealerweise folgen Sie den Namenskonventionen für Java-Pakete und tragen (in umgekehrter Reihenfolge) den Namen einer Ihnen gehörenden Internet-Domain gefolgt von einem Punkt und dem Namen der App ein. Verwenden Sie nur Kleinbuchstaben, und vermeiden Sie Sonderzeichen, insbesondere das Leerzeichen.

Auch die Angabe des Minimum SDK hat großen Einfluss auf die spätere Nutzbarkeit Ihres Programms. Damit legen Sie nämlich fest, welche Android-Version mindestens auf einem Gerät verfügbar sein muss, um Ihr Programm ausführen zu können. Beispielsweise ist erst ab Android 4 ein Zugriff auf Kalenderdaten über offizielle Schnittstellen möglich. Eine App, die diese nutzt, ist auf »älteren« Geräten mit *Gingerbread* oder gar *Cupcake* nicht lauffähig.

Wie Sie gleich sehen werden, gehören *Activities* zu den Grundbausteinen einer Android-Anwendung. Sofern das Häkchen vor Create Activity gesetzt ist (falls nicht, holen Sie es bitte nach), wird der Assistent automatisch eine Activity mit dem von Ihnen eingegebenen Namen erzeugen. Mit Finish schließen Sie den Projektassistenten. Nachdem Eclipse alle Dateien und Verzeichnisse angelegt hat, wird das Projekt automatisch im Package Explorer angezeigt. Denken Sie daran, dass Sie auf alle Eclipse-Sichten über das Menü Window • Show View zugreifen können.

2.1.2 Projektstruktur

Android-Apps bestehen aus einer ganzen Reihe von Artefakten, die zu einer baumartigen Struktur zusammengefasst werden. Klicken Sie die Wurzel dieses Baums (also den Projektnamen) oder eines seiner Unterelemente mit der rechten Maustaste an, öffnet sich ein Kontextmenü. Welche Funktionen hierbei angeboten werden, hängt vom angeklickten Objekt ab. Wie Sie bereits aus dem ersten Kapitel wissen, starten Sie auf diese Weise (Run As • Android Application) die Anwendung im Emulator oder einem angeschlossenen Gerät. Bitte probieren Sie dies nun aus.

Das gerade erzeugte Projekt enthält die in Abbildung 2.2 gezeigten Knoten *src*, *bin*, *gen*, *Android 4.0.3*, *assets* und *res*. Ferner sind die Dateien *proguard,cfg*, *AndroidManifest.xml* und *project.properties* (oder *default.properties*) zu sehen. Letztere wird von den Android Development Tools (ADT) verwendet und sollte von Ihnen nicht verändert werden. In ihr wird unter anderem vermerkt, gegen welche Plattform-Version eine Anwendung entwickelt wird. Auch der Knoten *Android 4.0.3* hat für Sie als Entwickler keine Bedeutung. Allerdings offenbart er die Pakete und Klassen, aus denen die Android-Klassenbibliothek besteht. Wenn Sie neugierig sind, können Sie einen Blick in seine weit verzweigte Struktur werfen.

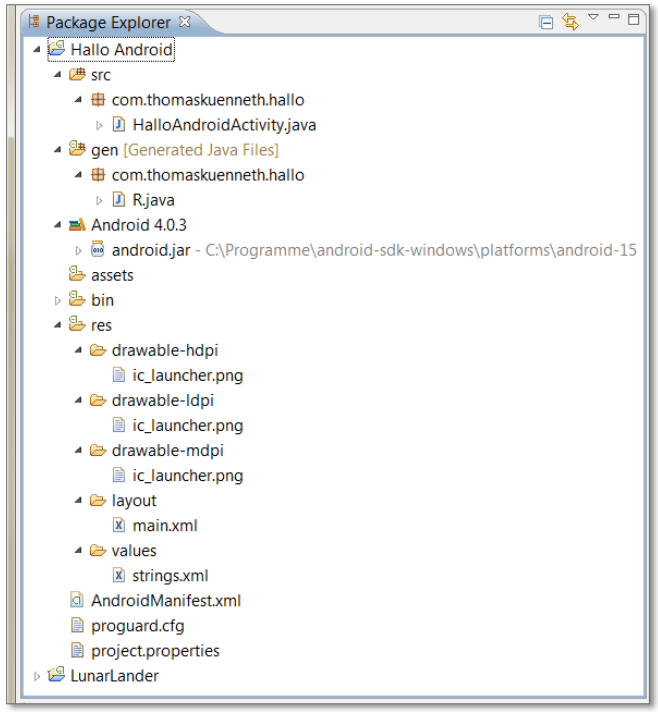


Abbildung 2.2 Das Projekt »Hallo Android« im Package Explorer

Sowohl *project.properties* als auch *Android 4.0.3* korrespondieren übrigens mit dem *Build Target* des Projektassistenten. Hätten Sie beim Anlegen des Projekts eine andere Android-Version gewählt, hätte auch der Knoten einen anderen Namen, beispielsweise *Android 1.6*.

Wenn Sie bei der Installation des Android SDK mehr als eine Plattform heruntergeladen haben, können Sie dies auch nachträglich ausprobieren. Klicken Sie hierzu im Package Explorer mit der rechten Maustaste auf den Projektnamen, und wählen Sie anschließend PROPERTIES. Klicken Sie nun auf ANDROID, und wählen Sie unter PROJECT BUILD TARGET eine andere Android-Version aus.

Da die ausgewählte Android-Version große Auswirkungen auf die zur Verfügung stehenden Klassen und Methoden hat, sollten Sie einen solchen Wechsel gut abwägen. Vor allem der Schritt von einer höheren zu einer niedrigeren Version kann zu Problemen führen, wenn beispielsweise auf Klassen Bezug genommen wird, die es in der früheren Version noch nicht gab. Das Projekt kann in diesem Fall nicht gebaut werden. Bevor Sie fortfahren, stellen Sie als BUILD TARGET bitte wieder ANDROID 4.0.3 ein, sofern Sie die Einstellung geändert haben.

Quelltexte werden in *src* abgelegt. Der Package Explorer zeigt aktuell nur die Klasse `HalloAndroidActivity`, die sich im Paket `com.thomaskuenneth.hallo` befindet. Beides hatten Sie im Projektassistenten eingetragen. Übrigens können Sie bequem neue Klassen anlegen, indem Sie das Paket mit der rechten Maustaste anklicken und NEW • CLASS anklicken.

Auch *gen* enthält ein Paket `com.thomaskuenneth.hallo`. Es beinhaltet die Klasse `R`. Wie Sie bald sehen werden, hat sie große Bedeutung beim Zugriff auf unterschiedlichste Elemente einer Android-App. Allerdings wird `R` durch die ADT verwaltet und darf deshalb von Ihnen nicht verändert werden. Ihr Inhalt ergibt sich aus Dateien des Verzeichnisses *res*. Dessen Unterordner *values* enthält beispielsweise die Datei *strings.xml*. Sie nimmt Texte auf, die später im Quelltext mithilfe der Klasse `R` referenziert werden. Wann immer Sie Änderungen an dieser Datei vornehmen, sorgen die Android Development Tools (ADT) automatisch dafür, dass `R` entsprechend angepasst und übersetzt wird.

Das Verzeichnis *assets* bleibt in vielen Projekten leer. Es nimmt nämlich Dateien auf, die zwar Bestandteil einer Anwendung sind, aber nicht durch Standardmechanismen von Android geladen werden können.

AndroidManifest.xml ist die zentrale Beschreibungsdatei einer Anwendung. In ihr werden unter anderem die Bestandteile des Programms aufgeführt. Wie Sie später noch sehen werden, sind dies sogenannte Activities, Services, Broadcast Receiver und Content Provider. Die Datei enthält aber auch Informationen darüber, welche Rechte eine App benötigt, welche Hardware sie erwartet und unter welchen Systemversionen sie lauffähig ist.

Um *AndroidManifest.xml* zu bearbeiten, stellen die ADT einen eigenen Editor zur Verfügung. Auf der in Abbildung 2.3 gezeigten Registerkarte APPLICATION werden Einstellungen vorgenommen, die sich auf die gesamte App beziehen. DEBUGGABLE beispielsweise steuert, ob eine Anwendung auf einem realen Gerät mit einem Debugger untersucht werden kann. Hinter der Registerkarte ANDROIDMANIFEST.XML verbirgt sich ein klassischer XML-Texteditor. Sie können die Datei also mittels Klapplisten, Text- und Ankreuzfeldern bearbeiten oder durch das Eingeben und Bearbeiten von XML-Tags und Attributen.

Öffnen Sie *AndroidManifest.xml*, und wechseln Sie in die XML-Ansicht, um sich einen ersten Eindruck von ihrer Struktur zu verschaffen. Es gibt ein Wurzelelement `<manifest>` mit den beiden Kindern `<application>` und `<uses-sdk>`. Android-Apps bestehen (neben den weiter oben bereits kurz genannten anderen Bausteinen) aus mindestens einer *Activity*. Hierbei handelt es sich, stark vereinfacht ausgedrückt, um Bildschirmseiten. Verschiedene Aspekte einer Anwendung wie Listen, Übersichten, Such- und Eingabemasken werden als eigene Activities realisiert und als Unterelemente von `<application>` in *AndroidManifest.xml* eingetragen.

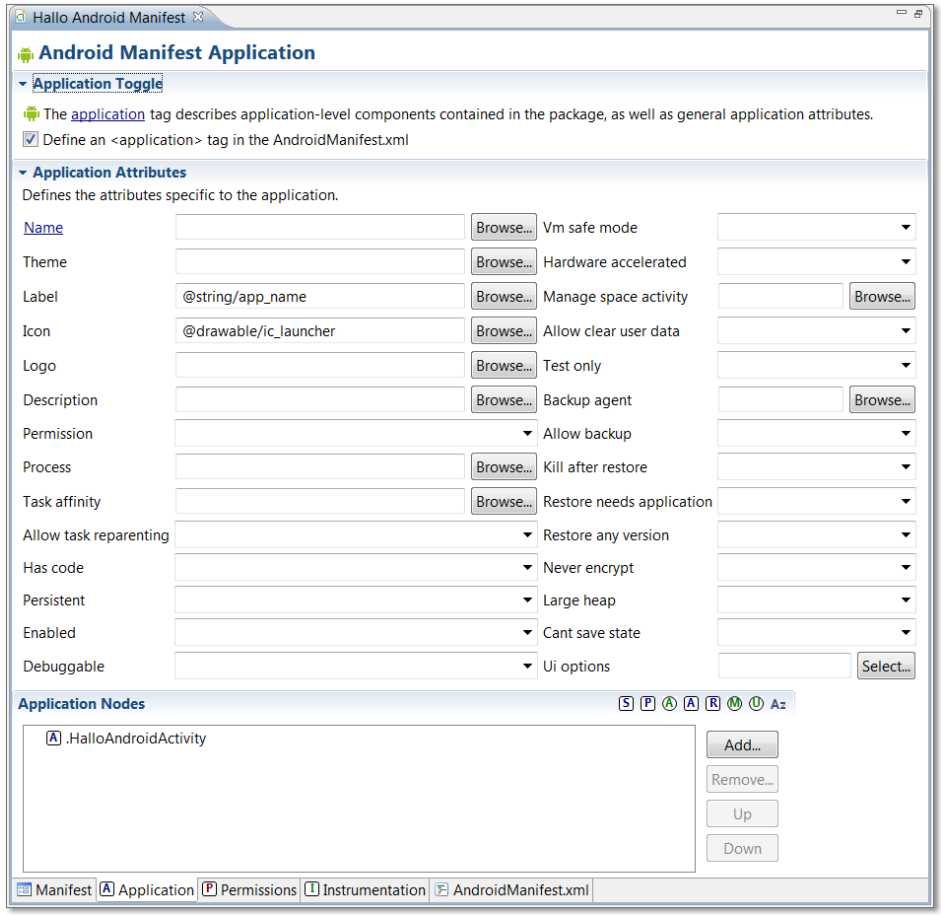


Abbildung 2.3 Editor für die Datei AndroidManifest.xml

Das Attribut `android:minSdkVersion` gibt an, welche Android-Version auf einem Gerät mindestens vorhanden sein muss, um die App nutzen zu können. Ist diese Voraussetzung nicht erfüllt, wird die Installation abgebrochen. Google Play zeigt das Programm in so einem Fall allerdings gar nicht an.

Bislang haben Sie nur mit einem virtuellen Android-Gerät (AVD) gearbeitet. Im folgenden Abschnitt zeige ich Ihnen, wie Sie mithilfe der sogenannten *Run Configurations* von Eclipse auch bei mehreren Emulatoren den Überblick behalten.

2.1.3 Run Configurations

Möchten Sie eine Anwendung nicht ausschließlich für den Eigengebrauch entwickeln, sollten Sie sie unter verschiedenen Android-Versionen und Gerätetypen testen. Falls Sie planen, Ihre App in Google Play zu veröffentlichen, wird dies sogar noch

viel wichtiger. Mit dem Ihnen bereits bekannten AVD Manager ist ein virtuelles Android-Gerät (AVD) mit wenigen Mausklicks angelegt. Zunächst laden wir aber eine weitere Plattform herunter. Wählen Sie hierzu Window • Android SDK Manager. Anschließend markieren Sie in der Liste Packages den Eintrag Android 1.5 (API 3) und klicken dann auf Install packages.

Nach dem Download können Sie den SDK Manager beenden. Nun steht dem Anlegen eines neuen virtuellen Geräts nichts mehr im Wege. Wählen Sie hierzu Window • AVD Manager. Mit New öffnen Sie den Dialog Create new Android Virtual Device (AVD). Übernehmen Sie die Einstellungen aus Abbildung 2.4. Sie erzeugen damit ein virtuelles Gerät, das unter Android 1.5 läuft. Allerdings hat es eine recht geringe Auflösung und wird standardmäßig im Quermodus betrieben. Um das Projekt *Hallo Android* in dem neu angelegten AVD auszuführen, klicken Sie den Projektnamen im Package Explorer mit der rechten Maustaste an und wählen Run As • Run Configurations.

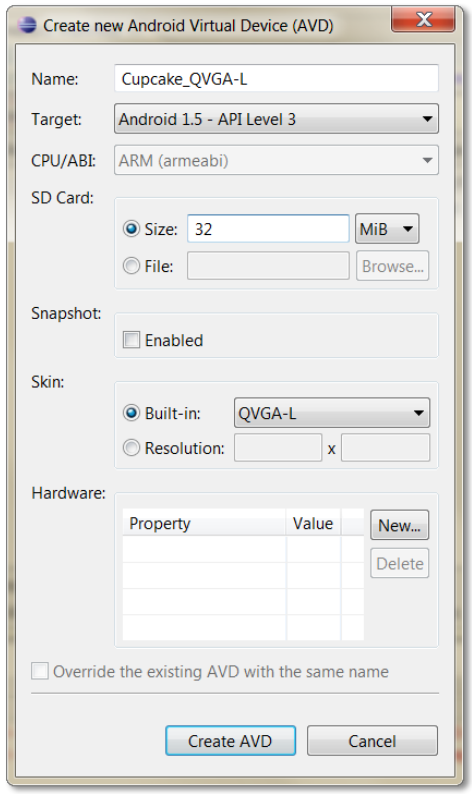


Abbildung 2.4 Dialog zum Anlegen eines AVDs

Als Sie die App das erste Mal im Emulator ausgeführt haben, hat Eclipse automatisch eine Run Configuration erzeugt. Sie ist in Abbildung 2.5 zu sehen. Auf der Registerkarte TARGET können Sie einstellen, ob die IDE selbstständig ein (virtuelles) Gerät auswählen oder während des Startvorgangs nachfragen soll. Klicken Sie auf MANUAL. Mit APPLY übernehmen Sie Ihre Änderung. Schließen Sie den Dialog mit CLOSE.

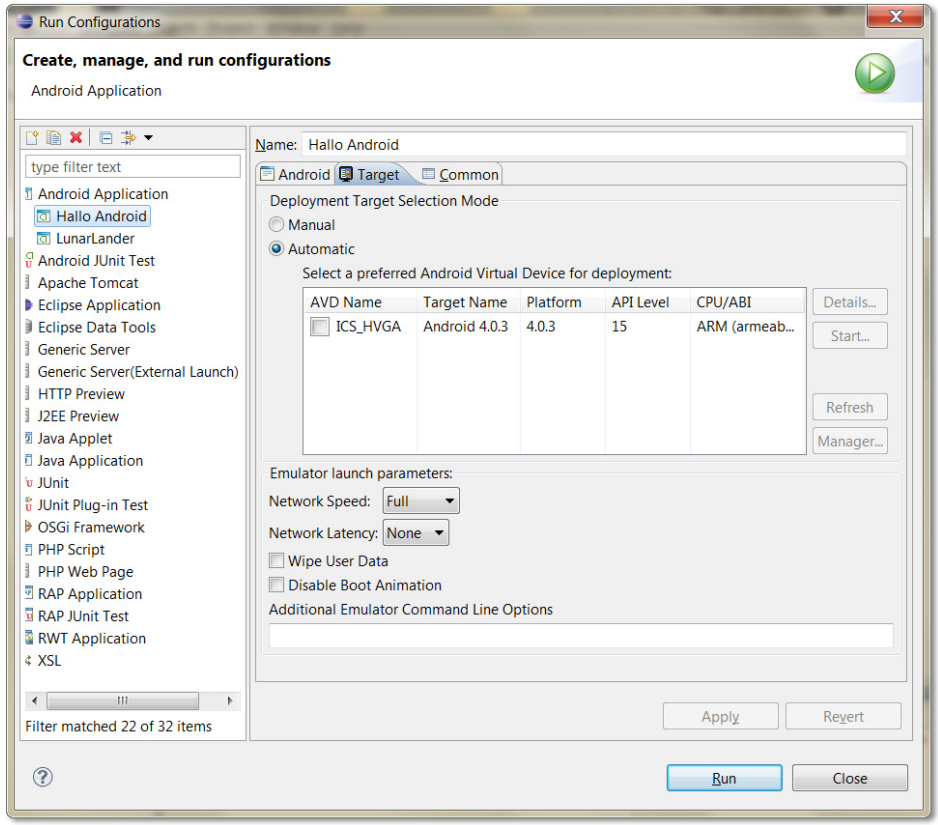


Abbildung 2.5 Der Dialog »Run Configurations«

Ist Ihnen aufgefallen, dass das von Ihnen angelegte Gerät mit Android 1.5 nicht unter SELECT A PREFERRED ANDROID VIRTUAL DEVICE FOR DEPLOYMENT erscheint? Der Grund ist, dass derzeit ANDROID 4.0.3 als PROJECT BUILD TARGET auf der Seite ANDROID der Projekteigenschaften eingetragen ist. Eclipse erkennt also, dass die Versionsnummer des AVDs zu niedrig ist. Bitte lassen Sie diese Einstellung zunächst bestehen.

Versuchen Sie nun, *Hallo Android* mit RUN • RUN zu starten. Eclipse öffnet den in Abbildung 2.6 gezeigten ANDROID DEVICE CHOOSER. Klicken Sie auf LAUNCH A NEW ANDROID VIRTUAL DEVICE, und wählen Sie CUPCAKE_QVGA-L, also das gerade angelegte virtuelle Gerät.

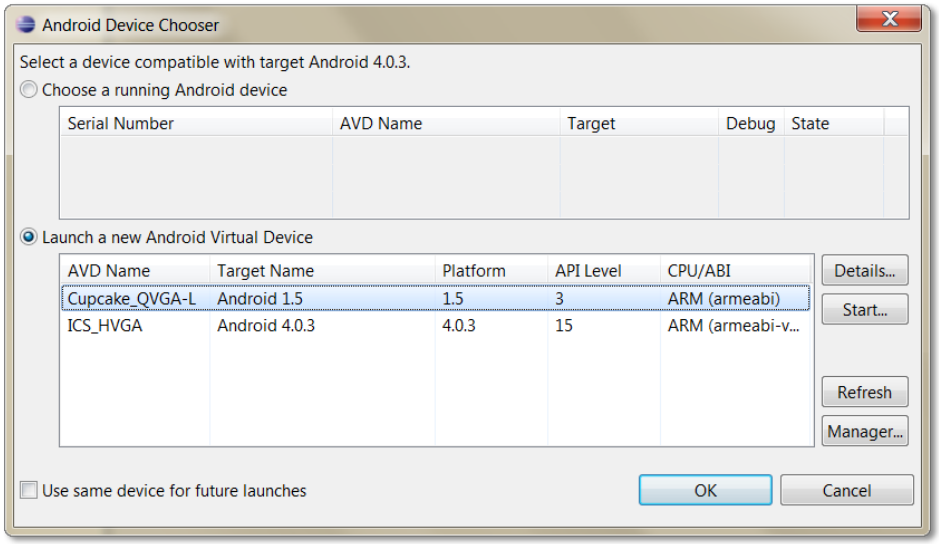


Abbildung 2.6 Der Dialog »Android Device Chooser«

Ein Klick auf OK startet den Emulator. Anschließend wird versucht, die App zu installieren. Dies schlägt fehl. Der Grund ist natürlich die zu niedrige Versionsnummer des AVDs. Eine entsprechende Fehlermeldung ist in der Sicht CONSOLE zu sehen.

Das Problem lässt sich allerdings sehr leicht beheben. Ändern Sie hierzu einfach in der Datei *AndroidManifest.xml* unter `android:minSdkVersion` die Zahl 15 in eine 3 um. Führen Sie anschließend RUN • RUN erneut aus. Da bereits eine Emulator-Instanz läuft, können Sie im ANDROID DEVICE CHOOSER auf CHOOSE A RUNNING ANDROID DEVICE klicken.

Im nächsten Abschnitt werden Sie erste Erweiterungen an *Hallo Android* vornehmen. Zunächst werde ich Ihnen zeigen, wie in Android Texte gespeichert werden und wie man in einer App auf diese zugreift.

2.2 Die Benutzeroberfläche

Die Benutzeroberfläche ist das Aushängeschild einer Anwendung. Gerade auf mobilen Geräten mit vergleichsweise kleinen Bildschirmen sollte jede Funktion leicht zugänglich und intuitiv erfassbar sein. Android unterstützt Sie bei der Gestaltung durch eine große Auswahl an Bedienelementen.

2.2.1 Texte

Bilder und Symbole sind ein wichtiges Gestaltungsmittel. Sinnvoll eingesetzt, helfen sie dem Anwender nicht nur beim Bedienen des Programms, sondern sorgen zudem für ein angenehmes, schönes Äußeres. Dennoch spielen auch Texte eine sehr wichtige Rolle. Sie werden in den unterschiedlichsten Bereichen einer Anwendung eingesetzt:

- als Beschriftungen von Bedienelementen
- für erläuternde Texte, die durch einen Screenreader vorgelesen werden (in Google Play stehen entsprechende Programme zum Download bereit)
- für Hinweis- und Statusmeldungen

Die fertige Version von *Hallo Android* soll den Benutzer zunächst begrüßen und ihn nach seinem Namen fragen. Im Anschluss wird ein persönlicher Gruß angezeigt. Nach dem Anklicken einer Schaltfläche beendet sich die App. Aus dieser Beschreibung ergeben sich die folgenden Texte. Die Bezeichner vor dem jeweiligen Text werden Sie später im Programm wiederfinden:

- `willkommen` – *Guten Tag. Schön, dass Sie mich gestartet haben. Bitte verraten Sie mir Ihren Namen.*
- `weiter` – *Weiter*
- `hallo` – *Hallo <Platzhalter>. Ich freue mich, Sie kennenzulernen*.
- `fertig` – *Fertig*

Ein Großteil der Texte wird zur Laufzeit so ausgegeben, wie sie schon während der Programmierung erfasst wurden. Eine kleine Ausnahme bildet die Grußformel. Sie besteht aus einem konstanten und einem variablen Teil. Letzterer ergibt sich erst, nachdem der Anwender seinen Namen eingetippt hat. Wie Sie gleich sehen werden, ist es in Android sehr einfach, dies zu realisieren.

Da Sie Apps in der Programmiersprache Java schreiben, könnten Sie die auszugebenden Meldungen einfach im Quelltext ablegen. Das sähe folgendermaßen aus (ich habe die String-Konstante aus Gründen der besseren Lesbarkeit in drei Teile zerlegt):

```
nachricht
   .setText("Guten Tag. Schön, dass " +
         "Sie mich gestartet haben." +
         " Bitte verraten Sie mir Ihren Namen.");
```

Das hätte allerdings eine ganze Reihe von Nachteilen. Da jede Klasse in einer eigenen Datei abgelegt wird, merkt man oft nicht, wenn man gleiche Texte mehrfach definiert. Dies vergrößert die Installationsdatei der App und kostet unnötig Speicher. Außerdem wird es auf diese Weise sehr schwer, mehrsprachige Anwendungen zu

bauen. Wenn Sie eine App über Google Play vertreiben möchten, sollten Sie neben den deutschsprachigen Texten aber mindestens eine englische Lokalisierung ausliefern.

Unter Android werden Texte daher zentral in der Datei *strings.xml* abgelegt. Sie befindet sich im Verzeichnis *values*. Ändern Sie die durch den Projektassistenten angelegte Fassung folgendermaßen ab:

```
<?xml version="1.0" encoding="utf-8"?>

<resources>

    <!-- Name der App -->
    <string name="app_name">Hallo Android!</string>

    <!-- Willkommensmeldung -->
    <string name="willkommen">
Guten Tag. Schön, dass Sie mich gestartet haben.
Bitte verraten Sie mir Ihren Namen.
    </string>

    <!-- persönlicher Gruß -->
    <string name="hallo">
Hallo %1$s. Ich freue mich, Sie kennenzulernen.
    </string>

    <!-- Beschriftungen für Schaltflächen -->
    <string name="weiter">Weiter</string>
    <string name="fertig">Fertig</string>

</resources>
```

Listing 2.1 strings.xml

Das Attribut `name` des Elements `<string>` wird später im Quelltext als Bezeichner verwendet. Der Name muss also projektweit eindeutig sein. Ist Ihnen im Listing die fett gesetzte Zeichenfolge `%1$s` aufgefallen? Android wird an dieser Stelle den vom Benutzer eingegebenen Namen einfügen. Wie dies funktioniert, zeige ich Ihnen später.

Hinweis

Die Zeilen `Hallo %1$s…` und `Guten Tag.` sind nicht eingerückt, weil die führenden Leerzeichen sonst in die App übernommen werden, was in der Regel nicht gewünscht ist.

Nach dem Speichern Ihrer Änderungen sollten Sie zunächst aber noch einen Blick auf die Klasse `R` unter *gen* werfen. Sie enthält zahlreiche als `public static final` gekennzeichnete Klassen, unter anderem `string`. Diese wiederum besteht aus einer Reihe von Konstanten des Typs `int`. Ihre Namen entsprechen den `name`-Attributen aus *strings.xml*.

Die Android Development Tools (ADT) aktualisieren `R` nach Änderungen an Dateien unterhalb des Verzeichnisses *res* automatisch. Wenn Sie also Texte oder Zeichenketten in Ihrer App benötigen, tragen Sie diese `<string>`-Elemente in *strings.xml* ein und greifen im Quelltext via `R.string` auf den jeweiligen Namen zu.

Vielleicht fragen Sie sich, wie Sie Ihr Programm mehrsprachig ausliefern können, wenn es genau eine zentrale *strings.xml* gibt. Neben dem Verzeichnis *values* kann es lokalisierte Ausprägungen geben, die auf das Minuszeichen und ein aus zwei Buchstaben bestehendes Sprachkürzel enden (zum Beispiel *values-en* oder *values-fr*).

Die Datei *string.xml* in diesen Ordnern enthält Texte in den korrespondierenden Sprachen, also Englisch oder Französisch. Muss Android auf eine Zeichenkette zugreifen, geht das System vom Speziellen zum Allgemeinen. Ist die Standardsprache also beispielsweise Englisch, wird zuerst versucht, den Text in *values-en/strings.xml* zu finden. Gelingt dies nicht, findet *values/strings.xml* Verwendung.

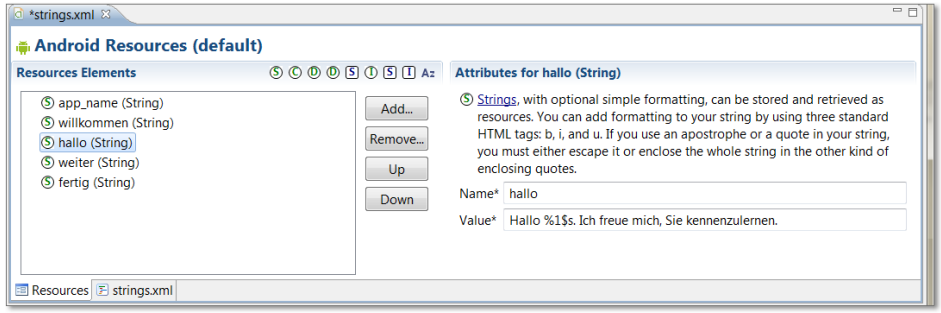


Abbildung 2.7 Die Registerkarte »Resources« des strings.xml-Editors

In dieser Datei müssen also alle Strings definiert werden. Lokalisierungen hingegen können unvollständig sein. Weitere Hinweise zur Internationalisierung und der Verwendung von Ressourcen finden Sie in der Entwicklerdokumentation unter *Localization*.[1]

Wie *AndroidManifest.xml* kann auch *strings.xml* mittels Listen und Schaltflächen bearbeitet werden. Die hierfür zuständige Registerkarte RESOURCES des Editors ist in Abbildung 2.7 zu sehen. Neben ADD und REMOVE zum Anlegen bzw. Löschen von Ele-

1 *http://developer.android.com/guide/topics/resources/localization.html*

menten gibt es die Schaltflächen UP und DOWN. Sie verschieben den markierten Eintrag in Richtung Dateianfang bzw. -ende.

Im folgenden Abschnitt stelle ich Ihnen sogenannte Views vor. Hierbei handelt es sich um die Grundbausteine, aus denen die Benutzeroberfläche einer App zusammengesetzt wird.

2.2.2 Views

Hallo Welt besteht auch nach vollständiger Realisierung aus sehr wenigen Bedienelementen:

- einem nicht editierbaren Textfeld, das den Gruß unmittelbar nach dem Programmstart sowie nach Eingabe des Namens darstellt
- einer Schaltfläche, die je nach Situation mit WEITER oder FERTIG beschriftet ist
- einem Eingabefeld, das nach dem Anklicken der Schaltfläche WEITER ausgeblendet wird

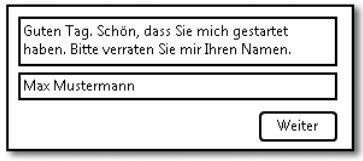


Abbildung 2.8 Prototyp der Benutzeroberfläche von »Hallo Android«

Wie die Komponenten auf dem Bildschirm platziert werden sollen, zeigt ein sogenannter *Wireframe*, den Sie in Abbildung 2.8 sehen. Man verwendet solche abstrakten Darstellungen gerne, um die logische Struktur einer Bedienoberfläche in das Zentrum des Interesses zu rücken.

Unter Android leiten alle Bedienelemente direkt oder indirekt von der Klasse `android.view.View` ab. Jede View belegt einen rechteckigen Bereich des Bildschirms. Ihre Position und Größe wird durch Layouts bestimmt. Diese wiederum erben von `android.view.ViewGroup`, ebenfalls ein Kind von `View`. Sie haben keine eigene grafische Repräsentation, sondern sind Container für weitere Views und ViewGroups.

Die Text- und Eingabefelder sowie Schaltflächen, die in *Hallo Android* verwendet werden, sind also Views. Konkret verwenden wir die Klassen `Button`, `TextView` und `EditText`. Wo sie auf dem Bildschirm positioniert werden und wie groß sie sind, wird hingegen durch die ViewGroup `LinearLayout` festgelegt.

Zur Laufzeit einer Anwendung manifestiert sich ihre Benutzeroberfläche demnach als Objektbaum. Aber nach welcher Regel wird er erzeugt? Wie definieren Sie als Entwickler den Zusammenhang zwischen einem Layout, einem Textfeld und einer

Schaltfläche? Java-Programmierer sind gewohnt, die Oberfläche programmatisch zusammenzusetzen. Nicht nur im Swing-Umfeld finden sich unzählige Ausdrücke im Stil von

```
JPanel p = new JPanel();
JButton b = new JButton();
p.add(b);.
```

Auch unter Android könnten Sie die Bedienelemente auf diese Weise zusammenfügen:

```
ScrollView v = new ScrollView(context);
LinearLayout layout = new LinearLayout(context);
layout.setOrientation(LinearLayout.VERTICAL);
v.addView(layout);
layout.addView(getCheckbox(context, Locale.GERMANY));
layout.addView(getCheckbox(context, Locale.US));
layout.addView(getCheckbox(context, Locale.FRANCE));
```

Listing 2.2 Beispiel für den programmatischen Bau einer Oberfläche

Allerdings ist dies nicht die typische Vorgehensweise – diese lernen Sie im folgenden Abschnitt kennen.

2.2.3 Oberflächenbeschreibungen

Eine Android-Anwendung beschreibt ihre Benutzeroberflächen normalerweise mittels XML-basierter Layoutdateien. Diese werden zur Laufzeit der App zu Objektbäumen »aufgeblasen«. Alle Bedienelemente von *Hallo Android* werden in einen Container des Typs `LinearLayout` gepackt. Seine Kinder erscheinen entweder neben- oder untereinander auf dem Bildschirm. Wie Sie gleich sehen werden, steuert das Attribut `android:orientation` die Laufrichtung. Auch die Größe der Views und ViewGroups wird so definiert. Hierfür gibt es `android:layout_width` und `android:layout_height`.

Oberflächenbeschreibungen werden in *layout*, einem Unterverzeichnis von *res*, gespeichert. Beim Anlegen des Projekts hat Eclipse dort die Datei *main.xml* abgelegt. Öffnen Sie diese, und ändern Sie sie folgendermaßen ab:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
>

    <TextView
      android:id="@+id/nachricht"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    />

    <EditText
      android:id="@+id/eingabe"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    />

    <Button
      android:id="@+id/weiter_fertig"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="right"
    />

</LinearLayout>
```

Listing 2.3 main.xml

Die XML-Datei bildet die Hierarchie der Benutzeroberfläche ab. Demzufolge ist `<LinearLayout>` das Wurzelelement. Mein Beispiel enthält die drei Kinder `<TextView>`, `<EditText>` und `<Button>`. Jedes Element hat die bereits kurz angesprochenen Attribute `android:layout_width` und `android:layout_height`.

Deren Wert `fill_parent` besagt, dass die Komponente die Breite oder Höhe des Elternobjekts erben soll. `wrap_content` hingegen bedeutet, dass sich die Größe aus dem Inhalt der View ergibt, beispielsweise der Beschriftung einer Schaltfläche. Die Zeile `android:layout_gravity="right"` sorgt dafür, dass die Schaltfläche rechtsbündig angeordnet wird.

Ist Ihnen aufgefallen, dass keinem Bedienelement ein Text oder eine Beschriftung zugewiesen wird? Und was bedeuten Zeilen, die mit `android:id="@+id/` beginnen? Wie Sie bereits wissen, erzeugt Android zur Laufzeit einer Anwendung aus den Oberflächenbeschreibungen entsprechende Objektbäume. Zu der in der XML-Datei spezifizierten Schaltfläche gibt es also eine Instanz der Klasse `Button`.

Um eine Referenz auf diese ermitteln zu können, wird ein Name definiert, beispielsweise `weiter_fertig`. Wie schon bei *strings.xml* sorgen die Android Development

Tools (ADT) dafür, dass nach Änderungen an Layoutdateien korrespondierende Einträge in der Klasse `R` vorgenommen werden.

Wenn Sie sich diese nach dem Speichern von *main.xml* ansehen, entdecken Sie die Klasse `id` mit den drei Konstanten `eingabe`, `nachricht` und `weiter_fertig`. Die Angabe `android:id="@+id/` sorgt also dafür, dass einer View ein Bezeichner zugewiesen wird, auf den mittels `R.id` zugegriffen werden kann. Wozu Sie das benötigen, zeige ich Ihnen im folgenden Abschnitt.

2.3 Programmlogik und -ablauf

Viele Desktop-Anwendungen sind datei- oder dokumentenzentriert. Egal ob Textverarbeitung, Tabellenkalkulation oder Layoutprogramm – ihr Aufbau ist stets gleich. Den überwiegenden Teil des Bildschirms oder Fensters belegt ein Arbeitsbereich, der ein Dokument (oder einen Teil davon) darstellt. Um ihn gruppieren sich Symbolleisten und Paletten, mit denen Elemente des Dokuments bearbeitet bzw. verändert werden.

Das gleichzeitige Darstellen von Werkzeugen und Inhalt ist auf den kleinen Bildschirmen mobiler Geräte aber nicht sinnvoll. Der Benutzer würde kaum etwas erkennen. Als Entwickler müssen Sie Ihre Anwendung deshalb in Funktionsblöcke oder Bereiche unterteilen, die genau einen Aspekt Ihres Programms abbilden.

Ein anderes Beispiel: E-Mail-Clients zeigen die wichtigsten Informationen zu eingegangenen Nachrichten häufig in einer Liste an. Neben oder unter der Liste befindet sich ein Lesebereich, der das aktuell ausgewählte Element vollständig anzeigt. Auch dies lässt sich aufgrund des geringen Platzes auf Smartphones nicht sinnvoll realisieren. Stattdessen zeigen entsprechende Anwendungen dem Nutzer zunächst eine Übersicht (die Liste der eingegangenen Nachrichten) und verzweigen in eine Detailansicht, sobald eine Zeile der Liste angeklickt wird.

2.3.1 Activities

Unter Android ist das Zerlegen einer App in solche (aufgabenorientierten) Teile bzw. Funktionsblöcke ein grundlegendes Architekturmuster. Die gerade eben skizzierten Aufgaben bzw. »Aktivitäten« – *E-Mail auswählen* und *E-Mail anzeigen* – werden zu Bausteinen, die die Plattform *Activities* nennt. Eine Anwendung besteht aus mindestens einer solchen Activity. Je nach Funktionsumfang können es aber auch viel mehr sein.

Normalerweise ist jeder Activity eine Benutzeroberfläche, also ein Baum bestehend aus Views und ViewGroups, zugeordnet. Activities bilden demnach die vom Anwender wahrgenommenen Bausteine einer App. Sie können sich gegenseitig aufrufen.

Die Vorwärtsnavigation innerhalb einer Anwendung wird beispielsweise auf diese Weise realisiert. Da das System Activities auf einem Stapel ablegt, müssen Sie sich als Entwickler nicht darum kümmern, von wem Ihre Activity aufgerufen wird. Drückt der Benutzer den ZURÜCK-Knopf, wird automatisch die zuvor angezeigte Activity reaktiviert.

Vielleicht fragen Sie sich, aus wie vielen Activities *Hallo Android* besteht. Theoretisch könnten Sie die App in drei Activities unterteilen, die Sie unabhängig voneinander anlegen müssten:

1. Begrüßung anzeigen
2. Namen eingeben
3. Personalisierten Gruß anzeigen

Das wäre sinnvoll, wenn die entsprechenden Aufgaben umfangreiche Benutzereingaben oder aufwendige Netzwerkkommunikation erfordern. Dies ist nicht der Fall. Da die gesamte Anwendung aus sehr wenigen Bedienelementen besteht, ist es in diesem Fall zielführender, alle Funktionen in einer Activity abzubilden.

Übernehmen Sie die im Folgenden dargestellte erste Version der Klasse `HalloAndroidActivity`. Um die Anwendung zu starten, wählen Sie RUN • RUN. Nach der Installation sollte das Emulatorfenster in etwa Abbildung 2.9 entsprechen.

Abbildung 2.9 Erste eigene Version von »Hallo Android«

Das Textfeld nimmt Eingaben entgegen. Das Anklicken der Schaltfläche WEITER löst aber selbstverständlich noch keine Aktion aus. Diese werden wir im nächsten Abschnitt implementieren. Zuvor möchte ich Sie aber mit einigen Schlüsselstellen des Quelltexts vertraut machen. Ganz wichtig: Jede Activity erbt von `android.app.Activity` oder von spezialisierten Kindklassen. Beispielsweise kennt die Plattform `ListActivity`[2], die das Erstellen von Auswahl- und Übersichtslisten stark vereinfacht.

```
package com.thomaskuenneth.hallo;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;

public class HalloAndroidActivity extends Activity {

   private TextView nachricht;
   private Button weiter_fertig;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      nachricht = (TextView) findViewById(R.id.nachricht);
      weiter_fertig = (Button) findViewById(R.id.weiter_fertig);

      nachricht.setText(R.string.willkommen);
      weiter_fertig.setText(R.string.weiter);
   }

}
```

Listing 2.4 Erste Version von HalloAndroidActivity.java

Haben Sie bemerkt, dass die gesamte Programmlogik innerhalb der Methode `onCreate()` liegt? Activities haben einen ausgeklügelten Lebenszyklus, den ich Ihnen in Kapitel 4, »Activities und Broadcast Receiver«, ausführlicher vorstelle. Seine einzelnen Stationen werden durch bestimmte Methoden der Klasse `Activity` realisiert, die Sie bei Bedarf überschreiben können. Beispielsweise informiert die Plattform eine Activity, kurz bevor sie beendet, unterbrochen oder zerstört wird. Die Methode

2 *http://developer.android.com/reference/android/app/ListActivity.html*

`onCreate()` wird immer überschrieben. Sie ist der ideale Ort, um die Benutzeroberfläche aufzubauen und um Variablen zu initialisieren.

Das Laden und Anzeigen der Bedienelemente reduziert sich auf eine Zeile Quelltext: `setContentView(R.layout.main);`. Sie sorgt dafür, dass alle Views und ViewGroups, die in der Datei *main.xml* definiert wurden, zu einem Objektbaum entfaltet werden und dieser als Inhaltsbereich der Activity gesetzt wird. Warum ich den Begriff »entfalten« verwende, erkläre ich Ihnen in Kapitel 5, »Benutzeroberflächen«.

Der Inhalt des Textfeldes `nachricht` und die Beschriftung der Schaltfläche `weiter_fertig` wird auf dieselbe Weise festgelegt. Zunächst ermitteln wir durch Aufruf der Methode `findViewById()` eine Referenz auf das gewünschte Objekt. Anschließend wird dessen `setText()` ein Text übergeben. Er ist in *values.xml* definiert.

2.3.2 Benutzereingaben

Um *Hallo Android* zu komplettieren, müssen wir auf das Anklicken der Schaltfläche `weiter_fertig` reagieren. Beim ersten Mal wird das Textfeld `eingabe` ausgelesen und als persönlicher Gruß in `nachricht` eingetragen. Anschließend wird das Textfeld ausgeblendet und die Beschriftung der Schaltfläche geändert. Wird diese ein zweites Mal angeklickt, beendet sich die App.

```
package com.thomaskuenneth.hallo;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class HalloAndroidActivity extends Activity {

   private TextView nachricht;
   private EditText eingabe;
   private Button weiter_fertig;

   private boolean erster_klick;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
```

```
        nachricht = (TextView) findViewById(R.id.nachricht);
        eingabe = (EditText) findViewById(R.id.eingabe);
        weiter_fertig = (Button) findViewById(R.id.weiter_fertig);

        erster_klick = true;

        nachricht.setText(R.string.willkommen);
        weiter_fertig.setText(R.string.weiter);

        weiter_fertig.setOnClickListener(new OnClickListener() {

           @Override
           public void onClick(View v) {
              if (erster_klick) {
                 nachricht.setText(getString(R.string.hallo,
                     eingabe.getText()));
                 eingabe.setVisibility(View.INVISIBLE);
                 weiter_fertig.setText(R.string.fertig);
                 erster_klick = false;
              } else {
                 finish();
              }
           }
        });
     }
}
```

Listing 2.5 Zweite Version von HalloAndroid.java

Um auf das Anklicken der Schaltfläche reagieren zu können, wird ein sogenannter *OnClickListener* registriert. Dieses Interface besteht aus der Methode `onClick()`. Die hier vorgestellte Implementierung nutzt die `boolean`-Variable `erster_klick`, um die durchzuführenden Aktionen zu bestimmen. `eingabe.setVisibility(View.INVISIBLE)`; blendet das Eingabefeld aus. `getString(R.string.hallo, eingabe.getText())` liefert den in *values.xml* definierten persönlichen Gruß und fügt an der Stelle `%1$s` den durch den Benutzer eingetippten Namen ein. Um die App zu beenden, wird die Methode `finish()` der Klasse `Activity` aufgerufen.

2.3.3 Der letzte Schliff

In diesem Abschnitt möchte ich Ihnen zeigen, wie Sie *Hallo Android* den letzten Schliff geben. Beispielsweise kann das System in leeren Eingabefeldern einen

Hinweis anzeigen, was der Benutzer eingeben soll. Hierzu fügen Sie in der Datei *strings.xml* die folgende Zeile ein:

```
<string name="vorname_nachname">Vorname Nachname</string>
```

Anschließend erweitern Sie in *main.xml* das Element `<EditText>` um das Attribut `android:hint="@string/vorname_nachname"`. Abbildung 2.10 zeigt das entsprechend abgeänderte Programm.

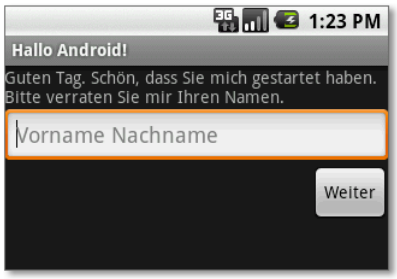


Abbildung 2.10 Leeres Eingabefeld mit Hinweis

Drücken Sie während der Eingabe eines Namens auf ↵, wandert der Cursor in die nächste Zeile. Auch die Höhe des Eingabefeldes nimmt zu. Dieses Verhalten lässt sich sehr leicht unterbinden. Erweitern Sie hierzu `<EditText>` einfach um `android:singleLine="true"`. Und noch ein Tipp: `android:inputType="textCapWords"` wandelt den ersten Buchstaben eines Worts automatisch in einen Großbuchstaben um.

Fällt Ihnen noch ein Defizit der gegenwärtigen Version auf? Solange der Benutzer keinen Namen eingetippt hat, sollte die Schaltfläche WEITER nicht anwählbar sein. Das lässt sich mithilfe eines sogenannten *TextWatchers* leicht realisieren. Fügen Sie in der Methode `onCreate()` vor der Zeile `erster_klick = true;` folgendes Quelltextfragment ein:

```
eingabe.addTextChangedListener(new TextWatcher() {

   @Override
   public void onTextChanged(CharSequence s, int start,
       int before,
       int count) {
   }

   @Override
   public void beforeTextChanged(CharSequence s, int start,
       int count,
       int after) {
   }
```

```
    @Override
    public void afterTextChanged(Editable s) {
       weiter_fertig.setEnabled(s.length() > 0);
    }
});
weiter_fertig.setEnabled(false);
```

Listing 2.6 Ausschnitt aus HalloAndroidActivity.java

Jedes Mal wenn ein Zeichen eingegeben oder gelöscht wird, ruft Android unsere Implementierung der Methode `afterTextChanged()` auf. Diese ist sehr einfach gehalten: Falls der Name mindestens ein Zeichen lang ist, kann die Schaltfläche WEITER angeklickt werden. Als kleine Übung können Sie versuchen, die Prüfroutine so zu erweitern, dass Vor- und Nachname vorhanden sein müssen. Prüfen Sie der Einfachheit halber, ob der eingegebene Text ein Leerzeichen enthält, das nicht am Anfang und nicht am Ende steht.

Kapitel 3
Von der Idee zur Veröffentlichung

Sie haben eine tolle Idee für eine App und würden am liebsten gleich loslegen? Ein bisschen Planung erleichtert nicht nur die Implementierung, sondern sorgt auch für zufriedene Nutzer. Warum das so ist, zeige ich Ihnen in diesem Kapitel.

Sie kennen das sicher, Sie haben eine Idee und möchten diese am liebsten sofort in die Tat umsetzen. Die Entwicklungsumgebung ist schnell gestartet. Erste Ergebnisse lassen sich unter Android in kurzer Zeit erzielen, wie das Beispiel im vorherigen Kapitel zeigt. Zunächst klappt das Erweitern eines so begonnenen Projekts noch recht gut. Irgendwann werden Sie aber feststellen, dass die Struktur der Anwendung nicht mehr so recht nachvollziehbar ist. Dann wird es auch zunehmend schwer, Änderungen durchzuführen und neue Funktionen einzubauen. In ein so aus dem Ruder gelaufenes Programm schleichen sich auch mehr und mehr Fehler ein. Und mancher Bug lässt sich nicht mehr entfernen, ohne die gesamte Konstruktion ins Wanken zu bringen.

Natürlich möchte ich Ihnen mit diesem Schreckensszenario nicht die Lust am Programmieren nehmen, im Gegenteil. Ganz gleich, ob Sie eine App nur für sich entwickeln oder in Google Play anbieten wollen – die Beschäftigung mit Googles offener Plattform soll Spaß machen. Deshalb ist es wichtig, die Stationen im Entstehungsprozess einer Anwendung zu kennen. Wenn Sie alle Schritte abgearbeitet (und gedanklich mit einem Häkchen versehen) haben, können Sie sicher sein, nichts Wesentliches vergessen zu haben. Vor allem aber lässt sich ihr Programm auch in Zukunft problemlos erweitern.

3.1 Konzept und Realisierung

Bevor Sie mit der Implementierung einer App beginnen, sollten Sie sich im Klaren darüber sein, was das Programm leisten und aus welchen Bausteinen oder Funktionsbereichen es bestehen wird. Widerstehen Sie der Versuchung, sich mit einer vagen Idee zufriedenzugeben. Sonst kann es leicht passieren, dass Sie Dinge implementieren, die schon im System vorhanden sind.

Wie Sie bereits wissen, bestehen Android-Anwendungen (zumindest aus der Sicht des Benutzers) aus Abfolgen von Aktivitäten, zum Beispiel *SMS eingeben*, *Kontakt auswählen*, *Foto aufnehmen* oder *Wähltastatur zeigen*. In der Konzeptionsphase legen Sie unter anderem fest, welche Aktivitäten Sie programmieren müssen und welche der bereits vorhandenen Sie gegebenenfalls einbinden können.

3.1.1 Konzeption

Für eine App ein Konzept zu schreiben, bedeutet keineswegs zwangsläufig, umfangreiche Dokumente zu erstellen. Beginnen Sie damit, den Zweck des Programms in wenigen Sätzen zu beschreiben. Wählen Sie die Formulierung so, dass jemand, der in Google Play zufällig über die Anwendung stolpert, Lust bekommt, sie herunterzuladen und auszuprobieren. Das könnte folgendermaßen aussehen:

> *Zeigt eine nach Kalendermonaten sortierte Liste der zwölf Sternzeichen an. Das Antippen eines Tierkreiszeichens verzweigt zu dem entsprechenden Eintrag in der Wikipedia.*

Diese beiden Sätze, ergänzt durch zwei Screenshots, genügen dem Google Play-Besucher, um zu entscheiden, ob er die App herunterladen möchte oder nicht. Wer kein Interesse an Astrologie hat, wird dies wahrscheinlich nicht tun. Auch für Sie als Entwickler sind die zwei Sätze ausreichend. Denn sie beschreiben den vollständigen Funktionsumfang des Programms. Als Nächstes leiten Sie aus dieser Beschreibung die groben Funktionsblöcke des Programms ab. Diese sind:

1. leere Liste bereitstellen
2. Liste mit Inhalt füllen
3. Webbrowser aufrufen

Da Listenansichten eine Kernfunktionalität der Android-Plattform sind, müssen Sie sich während der Konzeptphase nicht weiter um den ersten Punkt kümmern. Er wird erst im Verlauf der Implementierung interessant. Das Gleiche gilt für das Öffnen des Browsers. Der Inhalt der Liste hingegen repräsentiert die Fachlichkeit der App. Sie müssen deshalb zwei Fragen beantworten:

1. Was soll dargestellt werden?
2. Wie kommt es zustande?

An dieser Stelle ist es verführerisch, in Benutzeroberflächen zu denken. Sie sind aber erst später an der Reihe. Dass ein Listenelement also vielleicht das Symbol eines Sternzeichens enthält und dessen Namen ausgibt, spielt zunächst noch keine Rolle. Die Liste enthält alle existierenden Tierkreiszeichen, also genau zwölf. Diese sind nach Kalendermonaten in aufsteigender Reihenfolge sortiert. Tierkreiszeichen decken einen Datumsbereich ab. Sie haben demnach ein Start- und ein Enddatum.

Das aktuelle Sternzeichen lässt sich ermitteln, indem man prüft, in welchem Bereich das aktuelle Datum liegt.

3.1.2 Fachlogik

Kopieren Sie das Projekt *Tierkreiszeichen* aus dem Verzeichnis *Quelltexte* der Begleit-DVD des Buches in Ihren Eclipse-Arbeitsbereichsordner, und aktualisieren Sie anschließend den Arbeitsbereich mit File • Refresh. Die IDE zeigt dessen Speicherort im Untermenü File • Switch Workspace an. Um das Projekt im *Package Explorer* anzuzeigen, importieren Sie es mit File • Import. Klicken Sie im gleichnamigen Assistenten auf Existing Projects into Workspace, und folgen Sie den Anweisungen.

Wie Sie bereits wissen, können Sie mit `R.string` auf die Einträge zugreifen. Wir werden uns dies gleich zunutze machen, um den Tierkreis zusammenzusetzen. Zunächst möchte ich Ihnen aber noch einen Auszug der Klasse zeigen, die ein Tierkreiszeichen repräsentiert:

```
public final class Tierkreiszeichen {

   // Wann ein Sternzeichen beginnt
   private final int tag, monat, tierkreiszeichen;

   public Tierkreiszeichen(int tag, int monat,
                           int tierkreiszeichen) {
      this.tag = tag;
      this.monat = monat;
      this.tierkreiszeichen = tierkreiszeichen;
   }

   // Java-typische Getter und Setter

   public int getTag() {
      return tag;
   }

   public int getMonat() {
      return monat;
   }

   public int getTierkreiszeichen() {
      return tierkreiszeichen;
   }
```

```
    public String getName(Context context) {
       return context.getString(getTierkreiszeichen());
    }

    /**
     * Liefert einen Wert aus {@code R.drawable},
     * der für das Zeichnen des
     * Sternzeichens verwendet werden kann.
     * @return Wert aus {@code R.drawable}
     */
    public int getIdForDrawable() {
       switch (getTierkreiszeichen()) {
       case R.string.aquarius:
          return R.drawable.aquarius;
       case R.string.aries:
          return R.drawable.aries;
       case R.string.cancer:
          return R.drawable.cancer;
       case R.string.capricornus:
          return R.drawable.capricornus;
       case R.string.gemini:
          return R.drawable.gemini;
       case R.string.leo:
          return R.drawable.leo;
       case R.string.libra:
          return R.drawable.libra;
       case R.string.pisces:
          return R.drawable.pisces;
       case R.string.sagittarius:
          return R.drawable.sagittarius;
       case R.string.scorpius:
          return R.drawable.scorpius;
       case R.string.taurus:
          return R.drawable.taurus;
       case R.string.virgo:
          return R.drawable.virgo;
       }
       // ggf. ein Standardbild liefern
       return R.drawable.icon;
    }
}
```

Listing 3.1 Tierkreiszeichen.java

Sehen Sie sich nun die Datei *strings.xml* an. Sie weist jedem Tierkreiszeichen einen eindeutigen Namen zu:

```
<string name="aries">Widder</string>
<string name="taurus">Stier</string>
<string name="gemini">Zwillinge</string>
<string name="cancer">Krebs</string>
<string name="leo">Löwe</string>
<string name="virgo">Jungfrau</string>
<string name="libra">Waage</string>
<string name="scorpius">Skorpion</string>
<string name="sagittarius">Schütze</string>
<string name="capricornus">Steinbock</string>
<string name="aquarius">Wassermann</string>
<string name="pisces">Fische</string>
```

Listing 3.2 Ausschnitt aus strings.xml

Ein Tierkreiszeichen speichert Tag und Monat, an dem es beginnt. Vorgänger- und Nachfolgerbeziehungen werden in der Klasse `Zodiak` abgebildet.[1] Hierzu ein Beispiel: Schütze (sagittarius) beginnt am 23. November und folgt auf Skorpion, der am 22.11. endet:

```
put(Calendar.OCTOBER, new Tierkreiszeichen(
      24, Calendar.OCTOBER, R.string.scorpius));
put(Calendar.NOVEMBER, new Tierkreiszeichen(
      23, Calendar.NOVEMBER, R.string.sagittarius));
```

`Zodiak` legt die zwölf Tierkreiszeichen in einer `Hashtable` ab. Deren Schlüssel ist der Monat, in dem ein Sternzeichen beginnt. Um auf Tierkreiszeichen zuzugreifen, stehen zwei Methoden zur Verfügung:

```
public static Tierkreiszeichen getTierkreiszeichenFuerMonat
(int monat) {
   return INSTANCE.get(monat);
}
```

Listing 3.3 Ausschnitt aus Zodiak.java

Um möglichst wenig Speicher zu belegen, wurde die Klasse `Zodiak` als Singleton realisiert. Da sie von `java.util.Hashtable` abgeleitet ist, kann direkt auf die Methode `get()` ihrer Elternklasse zugegriffen werden.

1 Die Datumsangaben stellen Mittelwerte dar und beziehen sich auf ein Jahr mit 365 Tagen. Schaltjahre werden also nicht berücksichtigt.

Im nächsten Abschnitt zeige ich Ihnen, wie Sie die Klassen `Zodiak` und `Tierkreiszeichen` in eine Benutzeroberfläche integrieren.

3.1.3 Benutzeroberfläche

Bevor Sie mit der Programmierung der Benutzeroberfläche beginnen, müssen Sie das Konzeptdokument um eine entsprechende Beschreibung erweitern. Hierbei geht es nicht um eine möglichst genaue Vorwegnahme des späteren Bildschirminhalts. Überlegen Sie sich stattdessen, welche Informationen Sie anzeigen möchten, in welcher Beziehung diese zueinander stehen und wie sie logisch angeordnet werden.

Der Benutzer der App *Tierkreiszeichen* möchte sicher wissen, wann ein Sternzeichen beginnt und wann es endet. Außer dem Namen erwartet er noch ein Bild oder Symbol desselben. Mit diesen Informationen können Sie eine logische Darstellung der GUI entwerfen. Hierfür haben Sie zahlreiche Möglichkeiten.

Neben speziellen Wireframe-Editoren gibt es Bibliotheken für gängige Visualisierungsprogramme, wie etwa Microsofts Visio oder das auf dem Macintosh verbreitete Omnigraffle. Auch einfache Strichzeichnungen, sogenannte Scribbles, sind ein geeignetes Mittel. Welche Variante Sie wählen, ist letztlich eine Frage des persönlichen Geschmacks.

Abbildung 3.1 zeigt einen Wireframe der Hauptansicht. Systembestandteile, beispielsweise die Statuszeile sowie der Anwendungsname, wurden bewusst weggelassen.

Abbildung 3.1 Wireframe der Listenansicht

Android stellt mit `ListView` eine Komponente zur Verfügung, die auch mehrzeilige Elemente und Grafiken problemlos darstellen kann. Wie sie in *Tierkreiszeichen* eingesetzt wird, zeige ich im Folgenden. In *icon_text_text.xml* wird der Aufbau eines Listenelements definiert. *RelativeLayouts* beschreiben die Lage von Views in Relation zu anderen Bedienelementen.

Hierfür werden Positionsangaben wie `android:layout_below` oder `android:layout_toRightOf` verwendet. Entsprechende Oberflächenbeschreibungen sind nicht ganz so

leicht zu lesen, wie die Ihnen bereits bekannten LinearLayouts, benötigen aber zur Laufzeit weniger Speicher und werden schneller verarbeitet.


```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="?android:attr/listPreferredItemHeight"
   android:padding="6dip">

   <ImageView android:id="@+id/icon"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginRight="6dip"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent" />

   <TextView android:id="@+id/text1"
      android:textAppearance="?android:attr/textAppearanceMedium"
      android:layout_marginTop="4dip"
      android:layout_toRightOf="@id/icon"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />

   <TextView android:id="@+id/text2"
      android:textAppearance="?android:attr/textAppearanceSmall"
      android:layout_below="@id/text1"
      android:layout_alignLeft="@id/text1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />

</RelativeLayout>
```

Listing 3.4 icon_text_text.xml

Wie aus *icon_text_text.xml* ein Objektbaum wird, zeige ich Ihnen gleich. Zunächst aber möchte ich kurz demonstrieren, mit wie wenig Aufwand sich unter Android eine Listenanzeige realisieren lässt. Die Klasse `ListActivity` realisiert eine Listenansicht, die normalerweise den gesamten Bildschirm (natürlich mit Ausnahme der Statuszeile sowie der Fensterdekoration) ausfüllt.

Die Methode `getListView()` dieser Activity liefert eine Referenz auf ein zentrales Objekt des Typs `ListView`. Um auf das Antippen eines Eintrags zu reagieren, wird ein

sogenannter `OnItemClickListener` registriert. Die Beispielimplementierung nutzt dies, um den eingebauten Webbrowser mit einer bestimmten URL zu starten. Ausführliche Informationen zu sogenannten *Intents* finden Sie in Kapitel 4, »Activities und Broadcast Receiver«.

```
public class TierkreiszeichenActivity extends ListActivity
      implements OnItemClickListener {

   private TierkreiszeichenAdapter adapter;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // hier werden die Tierkreiszeichen gespeichert
      adapter = new TierkreiszeichenAdapter(this);
      setListAdapter(adapter);
      // auf das Antippen von Listenelementen reagieren
      getListView().setOnItemClickListener(this);
   }

   @Override
   public void onItemClick(AdapterView<?> parent,
         View view,
         int position,
         long id) {
      Tierkreiszeichen zeichen = (Tierkreiszeichen)
         adapter.getItem(position);
      String url = getString(R.string.wikipedia_url,
         zeichen.getName(this));
      // eine Webseite anzeigen
      Intent viewIntent =
         new Intent("android.intent.action.VIEW",
                    Uri.parse(url));
      startActivity(viewIntent);
   }
}
```

Listing 3.5 Ausschnitt aus TierkreiszeichenActivity.java

Welche Daten eine Liste anzeigt, wird durch sogenannte *Adapter* gesteuert. Android enthält eine ganze Reihe fertiger Klassen, die beispielsweise String-Arrays (`android.widget.ArrayAdapter<String>`) oder Tabellenzeilen einer Datenbank (`android.widget.SimpleCursorAdapter`) so umwandeln, dass eine `ListView` sie darstellen kann. Diese greift über die Methoden des Interfaces `android.widget.ListAdapter` auf einen

Adapter und die durch ihn bereitgestellten Daten zu. Wie eine solche Implementierung aussehen kann, zeigt die Klasse `TierkreiszeichenAdapter`.


```
public class TierkreiszeichenAdapter extends BaseAdapter {

   private final List<Tierkreiszeichen> zodiak;
   private final LayoutInflater inflator;
   private final DateFormat df;
   private final Calendar cal;

   public TierkreiszeichenAdapter(Context context) {
      // wird für das Aufblasen der XML-Datei benötigt
      inflator = LayoutInflater.from(context);
      // Tierkreiszeichen für alle Monate ermitteln
      zodiak = new ArrayList<Tierkreiszeichen>();
      for (int monat = Calendar.JANUARY;
               monat <= Calendar.DECEMBER; monat++) {
         Tierkreiszeichen zeichen = Zodiak
               .getTierkreiszeichenFuerMonat(monat);
         zodiak.add(zeichen);
      }
      // Legt fest, in welchem Format das Datum ausgegeben wird
      df = new SimpleDateFormat(context.getString(
               R.string.format_string));
      cal = Calendar.getInstance();
   }

   @Override
   public int getCount() {
      return zodiak.size();
   }

   @Override
   public Object getItem(int position) {
      return zodiak.get(position);
   }

   @Override
   public long getItemId(int position) {
      return position;
   }
```

```
@Override
public View getView(int position, View convertView,
   ViewGroup parent) {
   ViewHolder holder;

   // falls nötig, convertView bauen
   if (convertView == null) {
      // Layoutdatei entfalten
      convertView = inflator.inflate(R.layout.icon_text_text,
         parent, false);

      // Holder erzeugen
      holder = new ViewHolder();
      holder.name = (TextView) convertView
            .findViewById(R.id.text1);
      holder.datumsbereich = (TextView) convertView
            .findViewById(R.id.text2);
      holder.icon = (ImageView) convertView
            .findViewById(R.id.icon);

      convertView.setTag(holder);
   } else {
      // Holder bereits vorhanden
      holder = (ViewHolder) convertView.getTag();
   }

   Context context = parent.getContext();
   Tierkreiszeichen zeichen = (Tierkreiszeichen)
      getItem(position);
   holder.name.setText(zeichen.getName(context));
   holder.icon.setImageResource(zeichen.getIdForDrawable());
   cal.set(Calendar.DAY_OF_MONTH, zeichen.getTag());
   cal.set(Calendar.MONTH, zeichen.getMonat());
   String datum1 = df.format(cal.getTime());
   if (++position >= getCount()) {
      position = 0;
   }
   zeichen = (Tierkreiszeichen) getItem(position);
   cal.set(Calendar.DAY_OF_MONTH, zeichen.getTag() - 1);
   cal.set(Calendar.MONTH, zeichen.getMonat());
   String datum2 = df.format(cal.getTime());
   holder.datumsbereich.setText(context
         .getString(R.string.interval, datum1, datum2));
```

```
        return convertView;
    }

    static class ViewHolder {
        TextView name, datumsbereich;
        ImageView icon;
    }
}
```

Listing 3.6 TierkreiszeichenAdapter.java

Im Konstruktor werden die Tierkreiszeichen in einer `ArrayList` abgelegt und aufsteigend nach Monaten sortiert. Die Methode `getCount()` liefert die Länge der Liste (und damit die Zahl der insgesamt anzeigbaren Zeilen), `getItem()` das Element an einer bestimmten Position. Die Methode `getView()` baut aus den Daten des Modells einen Eintrag zusammen und stellt ihn in Gestalt einer `View`-Instanz zur Verfügung.

Aus Effizienzgründen puffert Android eine gewisse Menge solcher Objekte. Im Falle der erstmaligen Verwendung ist der Parameter `convertView` gleich `null`. Dann wird mithilfe eines `LayoutInflator` aus einer XML-Datei (*icon_text_text.xml*) ein entsprechender Komponentenbaum erzeugt und der Variable `convertView` zugewiesen.

Anschließend wird deren Methode `setTag()` ein sogenannter `ViewHolder` übergeben. Er fungiert als eine Art Platzhalter, um später einfach und effizient an die Elemente des Baums zu gelangen. Der erneute Aufruf der Methode `findViewById()` wäre wesentlich kostspieliger.

Im folgenden Abschnitt zeige ich Ihnen, wie Sie Ihr Programm mit Debug-Ausgaben versehen und wie Sie es auf die Veröffentlichung in Google Play vorbereiten.

3.2 Vom Programm zum Produkt

Eine Idee in Quelltext umzusetzen, ist für viele Entwickler der interessanteste Teil des Programmierens. Das Erstellen von Tests, das Suchen und Beheben von Fehlern hingegen wird oftmals als eher unangenehme Pflicht angesehen. Wenn Sie Ihre App anderen Personen zugänglich machen möchten, ist eine gewissenhafte Kontrolle aber unerlässlich. Andernfalls droht harsche Kritik. Ein Blick auf die Kommentare in Google Play offenbart, wie viel die Käufer bzw. Nutzer von den Programmierern erwarten.

3.2.1 Protokollierung

Allein schon aus Platzgründen kann dieses Buch leider keine Anleitung für das Schreiben von sauberem Code enthalten. Auch wie man Tests entwirft und einsetzt, lesen Sie bei Bedarf in entsprechender Spezialliteratur nach (im Anhang A finden Sie eine Lektüreliste). Aber wie Sie Android beim Aufspüren von Problemen unterstützt und wie Sie Ihre Apps auf echter Hardware und im Emulator testen, zeige ich Ihnen in diesem sowie dem folgenden Abschnitt.

Java-Entwickler verwenden zur schnellen Analyse oder Protokollierung gerne das Konstrukt `System.out.println()`. Dies funktioniert auch unter Android. Legen Sie ein neues Android-Projekt an (Sie können der ebenfalls zu erzeugenden Activity einen beliebigen Namen geben), und fügen Sie vor der schließenden Klammer des Methodenrumpfes von `onCreate()` dieses Quelltextfragment ein:

```
int fakultaet = 1;
System.out.println("0! = " + fakultaet);
for (int i = 1; i <= 5; i++) {
   fakultaet = i * fakultaet;
   System.out.println(i + "! = " + fakultaet);
}
```

Listing 3.7 Berechnung und Ausgabe der Fakultät

Sie finden das Projekt *DebugDemo* im Verzeichnis *Quelltexte* auf der Begleit-DVD. Wenn Sie die App im Emulator ausführen, sehen Sie die berechneten Fakultäten zunächst nicht. Diese werden in der in Abbildung 3.2 gezeigten Sicht LogCat angezeigt. Um sie zu öffnen, wählen Sie Window • Show View • Other. Es erscheint daraufhin der Dialog Show View. LogCat befindet sich unterhalb des Knotens Android.

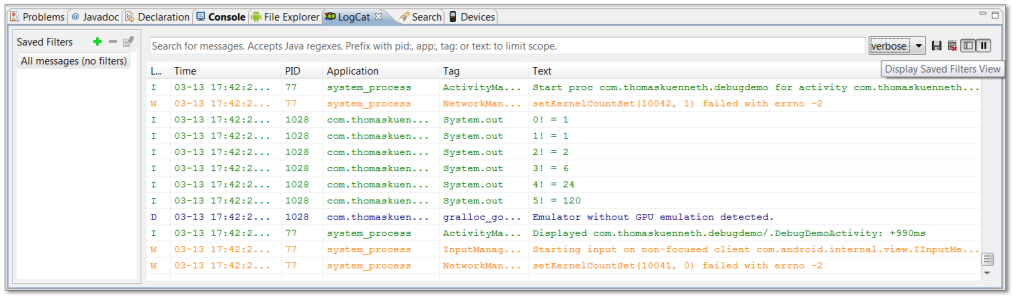


Abbildung 3.2 Die Sicht »LogCat«

Da LogCat nicht nur das Protokoll Ihrer Anwendung enthält, sondern gegebenenfalls auch Systemmeldungen anzeigt, wird die Darstellung schnell unübersichtlich. Aus diesem Grund können Sie die Ausgabe filtern. Öffnen Sie hierzu die Ansicht Saved Filters, indem Sie das zweigeteilte Rechteck im oberen rechten Randbereich von LogCat (Display Saved Filters View) anklicken. Wählen Sie anschließend Add a new logcat filter (das grüne Plus-Symbol), um den in Abbildung 3.3 gezeigten Dialog Log Filter aufzurufen.

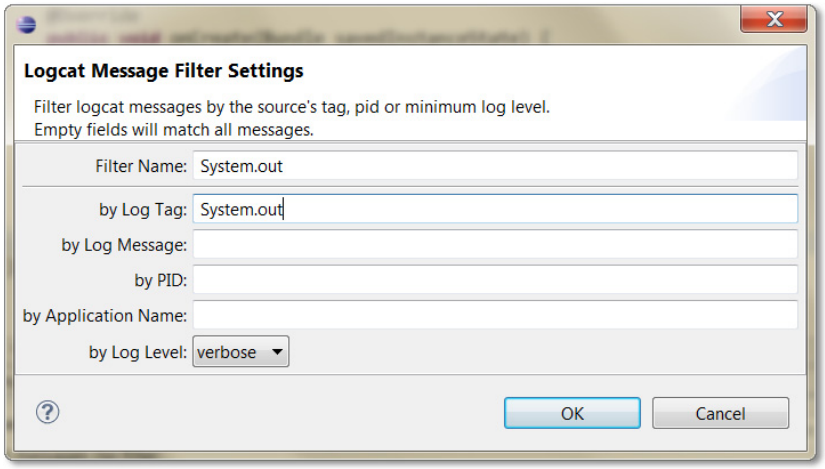


Abbildung 3.3 Dialog zum Anlegen und Bearbeiten von Filtern

Um nur noch die mit System.out markierten Zeilen anzeigen zu lassen, übernehmen Sie die Angaben aus Abbildung 3.3 und schließen den Dialog mit OK. Zwischen Filtern und der Anzeige aller Ausgaben schalten Sie bequem durch Anklicken der Einträge in Saved Filters um. Jeder Filter erscheint als eigene Registerkarte in LogCat. Mit Edit selected LogCat Filter und Delete selected logcat Filter können Sie bestehende Filter bearbeiten und löschen.

Haben Sie in Abbildung 3.2 die mit verbose vorbelegte Klappliste bemerkt? Sie enthält die Stufen (engl. *log level*) *verbose*, *debug*, *info*, *warn*, *error* und *assert*, die in gängigen Frameworks verwendet werden, um die Wichtigkeit bzw. den Schweregrad eines Protokolleintrags zu bestimmen.

Die Idee ist, für entsprechende Ausgaben nicht `System.out.println()` zu verwenden, sondern spezielle Logging-Methoden. Android stellt mit der Klasse `android.util.Log` eine besonders einfach zu handhabende Variante zur Verfügung. Deren statische Methoden `v()`, `d()`, `i()`, `w()` und `e()` repräsentieren die oben genannten log levels. Neben dem auszugebenden Text erwarten sie ein sogenanntes *Tag*. Es kennzeichnet die Quelle des Protokolleintrags, also im Allgemeinen die Klasse oder Activity.

Google empfiehlt, das Tag als Konstante zu definieren, zum Beispiel:

```
private static final String TAG = "MyActivity";
```

Um Tippfehler zu vermeiden, rate ich stattdessen zu

```
private static final String TAG = Tester.class.getSimpleName();
```

Anstelle von `Tester` verwenden Sie natürlich den Namen Ihrer Klasse. Übernehmen Sie die folgenden Anweisungen in Ihre Activity, und starten Sie danach die App:

```
Log.v(TAG,
    "ausführliche Protokollierung, nicht in Produktion verwenden");
Log.d(TAG, "Debug-Ausgaben");
Log.i(TAG, "Informationen");
Log.w(TAG, "Warnungen");
Log.e(TAG, "Fehler");
```

Listing 3.8 Erzeugen von Protokolleinträgen

Die fünf Ausgaben erscheinen in der Sicht LogCat. Wählen Sie ein Element aus der Klappliste aus, um Einträge mit niedrigerer Priorität auszublenden. Haben Sie beispielsweise INFO aktiviert, sind Aufrufe, die durch die Methoden `v()` (VERBOSE) und `d()` (DEBUG) erzeugt wurden, nicht zu sehen. Ein Klick auf VERBOSE zeigt alle Zeilen an.

Auf echter Hardware werden Ausgaben des log levels VERBOSE ignoriert, sofern die Anwendung aus einer sogenannten *.apk*-Datei installiert wurde. Alle anderen Stufen bleiben erhalten. Aus diesem Grund sollten Sie überlegen, welche Meldungen ausschließlich für die Entwicklung relevant sind. Diese können Sie der Methode `v()` übergeben. Für Warnungen und Fehler sind `w()` und `e()` gedacht. Debug-Ausgaben (`d()`) können unter bestimmten Umständen auch in produktiven Systemen sinnvoll sein. Deshalb rate ich Ihnen, entsprechende Methodenaufrufe mit einer `if()`-Abfrage zu kapseln. Definieren Sie hierzu eine Konstante

```
private static final boolean DEBUG = true;
```

Die Ausgabe sieht dann so aus:

```
if (DEBUG) {
   Log.d(TAG, "eine Debug-Ausgabe");
}
```

Listing 3.9 Steuerung einer Debug-Ausgabe mittels Konstante

Tipp

Bei umfangreicheren Projekten kann es sich lohnen, die Variable `DEBUG` in eine eigene Klasse auszulagern. Sie brauchen deren Wert dann nur an einer zentralen Stelle zu ändern. In diesem Fall muss der Zugriff aber natürlich jeder Klasse (`public`) gestattet werden.

Auf diese Weise können Sie bequem kontrollieren, ob Debug-Ausgaben erscheinen. Wenn Sie eine Exception gefangen haben, müssen Sie übrigens nicht mühselig einen passenden String zusammensetzen, sondern können sie als zusätzlichen Parameter an die Ihnen bereits bekannten fünf Methoden übergeben. Auch hierzu ein Beispiel:

```
String s = null;
try {
   Log.i(TAG, "s ist " + s.length() + " Zeichen lang");
} catch (Throwable tr) {
   Log.e(TAG, "Es ist ein Fehler aufgetreten.", tr);
} finally {
   Log.i(TAG, "s ist " + s);
}
```

Listing 3.10 Ausgeben eines Stacktrace im Fehlerfall

Da `s` mit `null` initialisiert wurde, ist eine `NullPointerException` unausweichlich. Sie wird aufgrund des `catch (Throwable tr)` gefangen und mittels `e()` als Fehler protokolliert. Wenn Sie eine Zeile des Stacktrace anklicken und im VIEW MENU des LOGCAT GO TO PROBLEM wählen, zeigt Eclipse die Methode an, in der die Ausnahme aufgetreten ist.

3.2.2 Fehler suchen und finden

Protokolldateien sind ein wichtiges Hilfsmittel bei der Analyse von Anwendungsproblemen. Allerdings können und sollen sie die klassische Fehlersuche mit dem Debugger nicht ersetzen. Wie Sie Bugs auf Quelltextebene zu Leibe rücken, zeige ich Ihnen nun. Legen Sie hierzu ein neues Projekt mit Activity an, und ersetzen Sie deren `onCreate()`-Methode durch das folgende Quelltextfragment:

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
   Log.i(TAG, "fib(5) = " + fib(5));
}
```

Listing 3.11 Die Methode onCreate()

Fügen Sie Ihrer Activity ferner diese Methode hinzu:

```
private int fib(int n) {
   int fib;
   switch (n) {
   case 0:
      fib = 0;
   case 1:
      fib = 1;
      break;
   default:
      fib = fib(n - 1) + fib(n - 2);
   }
   return fib;
}
```

Listing 3.12 Methode zur Berechnung einer beliebigen Fibonacci-Zahl

Die *Fibonacci-Folge* ist eine unendliche Folge von Zahlen, bei der sich die jeweils folgende Zahl durch Addition ihrer beiden Vorgänger ergibt. Für `n` größer oder gleich 2 gilt demnach: `fib(n) = fib(n – 1) + fib(n – 2)`. Für die beiden Spezialfälle 0 und 1 wurde festgelegt: `fib(0) = 0` und `fib(1) = 1`. Der Aufruf `fib(5)` müsste also 5 ergeben. Starten Sie die App, um dies zu überprüfen. Denken Sie daran, dass das berechnete Ergebnis nicht direkt im Emulator angezeigt wird, sondern in der Sicht LogCat.

Leider ist dort 8 zu lesen. Lassen Sie uns mithilfe der Einzelschrittabarbeitung herausfinden, was hier passiert. Fügen Sie der Datei *AndroidManifest.xml* im Element `<application>` das Attribut `android:debuggable="true"` hinzu. Setzen Sie anschließend einen sogenannten *Line breakpoint* in die Zeile `switch (n) {` der Methode `fib()`, indem Sie im Randbereich des Java-Editors in dieser Zeile mit der linken Maustaste doppelklicken.

Wenn der Haltepunkt angelegt wurde, erscheint an der entsprechenden Position ein ausgefüllter blauer Kreis. Berühren Sie ihn mit der Maus, um den in Abbildung 3.4 dargestellten Tooltip einzublenden. Die angezeigte Zeilennummer kann bei Ihnen geringfügig variieren. Jetzt können Sie die App debuggen. Klicken Sie hierzu im Package Explorer die Projektwurzel mit der rechten Maustaste an, und wählen Sie Debug As • Android Application. Nach kurzer Zeit wechselt Eclipse in die sogenannte *Debug-Perspektive* und hält die Programmausführung in der Zeile mit Haltepunkt an. Einen etwaigen Hinweisdialog (Confirm Perspective Switch) bestätigen Sie mit Yes.

Die Möglichkeiten, die der Eclipse-Debugger Ihnen bietet, sind viel zu umfassend, um sie in diesem Buch auch nur andeuten zu können. Wenn Sie sich intensiver mit der Materie befassen möchten, rate ich Ihnen zu entsprechender Spezialliteratur, die ich in der Lektüreliste aufgeführt habe.

Sie können sich sehr einfach den aktuellen Wert einer Variablen anzeigen lassen, indem Sie die Maus auf ihren Namen bewegen. Nach dem ersten Halt ist `n` gleich 5. Klicken Sie in der Sicht DEBUG auf STEP OVER oder drücken die Taste F6, um die nächste Anweisung auszuführen. Dies ist `fib = fib(n – 1) + fib(n – 2);`. Mit RESUME (F8) lassen Sie das Programm bis zum Erreichen des nächsten Haltepunkts ohne Unterbrechung weiterlaufen.

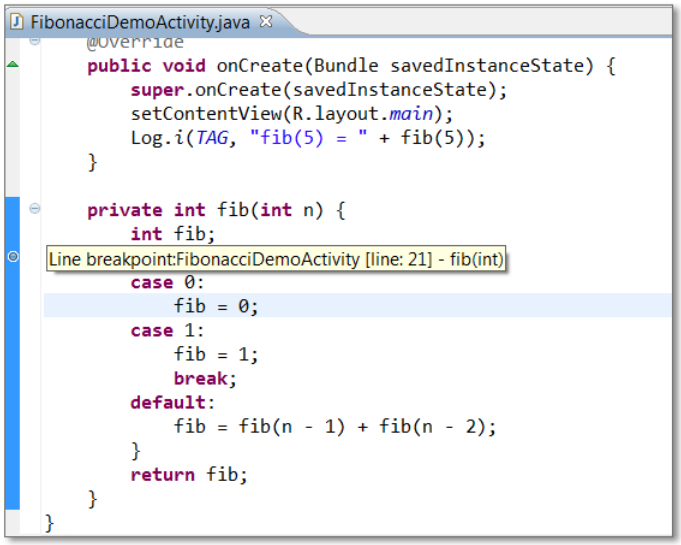


Abbildung 3.4 Ein Haltepunkt in der Methode »fib()«

Wiederholen Sie die Schritte mit STEP OVER und RESUME so lange, bis `n` den Wert 1 hat. Nun führt Sie ein STEP OVER in die Zeile `fib = 1;`. Dies ist erwartetes Verhalten. Sie können die App deshalb mit RESUME fortsetzen. Da 0 die letzte zu verarbeitende Zahl ist, müsste der Fehler jetzt auftreten. Nachdem das Programm durch den Debugger angehalten wurde, führt STEP OVER zur Zeile `fib = 1;`. Dies ist falsch, denn eigentlich hätte der Ausdruck `fib = 0;` ausgewertet werden müssen.

Wenn Sie die beiden `case`-Bereiche vergleichen, fällt sehr schnell das fehlende `break` auf. Dies erklärt das unerwartete Verhalten. Stoppen Sie den Debug-Vorgang, indem Sie in der Sicht DEBUG das Wurzelelement (den Namen Ihres Projekts) anklicken und anschließend TERMINATE auswählen. Fügen Sie `break;` in den Quelltext ein, und starten Sie die App erneut. In LOGVIEW wird daraufhin der richtige Wert 5 angezeigt.

3.2.3 Debuggen auf echter Hardware

Wenn Sie ein Android-Smartphone besitzen, können Sie die eben vorgestellte App direkt auf ihm debuggen. Gerade Programme, die Sie an andere weitergeben möchten, sollten Sie einem solchen Test unterziehen.

Unter Windows müssen Sie vor dem erstmaligen Anschließen des Geräts an den USB-Port Ihres Rechners möglicherweise einen aktualisierten Treiber installieren. Sie finden einen solchen im Unterordner *extras\google\usb_driver* des Android SDK-Installationsverzeichnisses. Falls Sie die Komponente noch nicht heruntergeladen haben, können Sie dies mit dem Android SDK Manager jederzeit nachholen. Informationen hierzu finden Sie in Abschnitt 1.3, »Entwicklungswerkzeuge«. Das Dokument *Google USB Driver*[2] enthält Installationsanleitungen für gängige Windows-Versionen und nennt die unterstützten Geräte.

Unter Mac OS X sind keine weiteren Vorkehrungen nötig. Als Linux-Nutzer beachten Sie bitte die Hinweise in Abschnitt *Setting up a Device for Development* des Dokuments *Developing on a Device*.[3]

Auf vielen Geräten müssen Sie jetzt noch das USB-Debugging aktivieren. Öffnen Sie gegebenenfalls die EINSTELLUNGEN, und tippen Sie anschließend auf ENTWICKLEROPTIONEN (siehe Abbildung 3.5). Setzen Sie ein Häkchen bei USB-DEBUGGING und kehren danach zum Startbildschirm zurück. Einige wenige Modelle (zum Beispiel das zum Zeitpunkt der Drucklegung in Deutschland nicht angebotene *Kindle Fire* von Amazon) sind ab Werk so eingestellt.

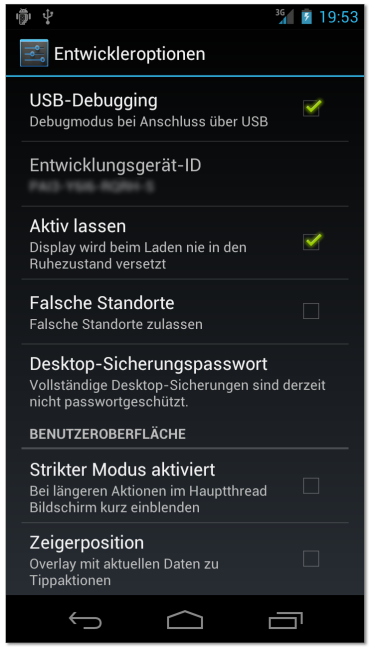


Abbildung 3.5 Die Seite »Entwicklung« der Android-Einstellungen

2 *http://developer.android.com/sdk/win-usb.html*
3 *http://developer.android.com/guide/developing/device.html#setting-up*

Starten Sie nun den Debug-Vorgang, indem Sie im Package Explorer die Projektwurzel mit der rechten Maustaste anklicken und DEBUG AS • ANDROID APPLICATION wählen. Falls Sie einen Emulator geöffnet haben, fragt Eclipse nach, ob die App dort oder auf dem angeschlossenen Gerät ausgeführt werden soll.

3.3 Anwendungen verteilen

Wenn Sie die eigentliche Entwicklung Ihres Programms abgeschlossen haben und die App alle Tests erfolgreich durchlaufen hat, können Sie die Veröffentlichung bzw. Verteilung vorbereiten. Was Sie hierbei beachten sollten, zeige ich Ihnen in diesem Abschnitt.

3.3.1 Verteilbare Anwendungen

Jede App hat ein Icon, das in *Google Play* neben dem Programm- und Herstellernamen angezeigt wird. Nach dem Herunterladen und Installieren tippt der Nutzer auf dieses Symbol, um die Anwendung zu starten. Das Icon ist also seine Visitenkarte. Entsprechend viel Aufwand sollten Sie in dessen Gestaltung investieren. Dies mag übertrieben wirken. Aber die Kommentare in Google Play enthalten unzählige Beschwerden über unästhetische Grafik.

Android erwartet Programm-Icons in unterschiedlichen Größen. Aus diesem Grund sollten Sie das Symbol mit einem Vektorgrafikprogramm gestalten und später als Bitmap in den benötigten Größen exportieren. Dies sind 36 × 36, 48 × 48, 72 × 72 und 96 × 96 Pixel. Beim Anlegen eines neuen Projekts werden im Ordner *res* automatisch die Unterverzeichnisse *drawable_ldpi*, *drawable_mdpi* und *drawable_hdpi* erzeugt und mit einem Standard-Icon in der jeweiligen Größe gefüllt. Grafiken mit 96 × 96 Pixel legen Sie unter *drawable_xhdpi* ab. Zum Zeitpunkt der Drucklegung erzeugen die Android Development Tools dieses Verzeichnis nicht automatisch.

Der Name der Dateien ist hierbei stets gleich. Er ergibt sich aus dem Wert des Attributs `android:icon` in *AndroidManifest.xml* (beispielsweise `android:icon="@drawable/icon"`). Richtlinien für die Gestaltung von Symbolen finden Sie im Dokument *Icon Design Guidelines*.[4] Es informiert Sie unter anderem darüber, aus welchen Farben Ihre Icons bestehen sollten und was Sie bei der Perspektive beachten müssen.

Damit sich Kunden über Ihr Programm informieren können, sollten Sie den Aufbau einer kleinen Seite im Internet in Erwägung ziehen. Entsprechende Hosting-Angebote kosten mittlerweile nur noch sehr wenig Geld. Neben der Möglichkeit, Werbung für Ihr Produkt zu machen, ist dies der ideale Ort für Frequently Asked Questions sowie

4 *http://developer.android.com/guide/practices/ui_guidelines/icon_design.html*

Tipps und Tricks. Die *Developer Console* von Google Play, die ich Ihnen später vorstelle, sieht die Eingabe von Kontaktdaten wie Website und E-Mail-Adresse ohnehin vor.

Jetzt haben Sie fast alle Stationen auf dem Weg zum fertigen Produkt absolviert. Sie müssen nur noch entscheiden, ob Sie Ihre App in Eigenregie vermarkten oder über Google Play bzw. einen der immer zahlreicher werdenden *Secondary Markets* vertreiben möchten. Was es damit auf sich hat, erfahren Sie in den folgenden Abschnitten.

Zuvor müssen Sie Ihr Programm aber digital signieren und als sogenanntes *Application Package* exportieren. Hierbei handelt es sich um eine komprimierte Installationsdatei mit der Endung *.apk*, die alle Bestandteile einer App (also Klassen, Ressourcen sowie sonstige Artefakte) bündelt. Sobald Sie eine Anwendung aus der IDE heraus starten, generieren die *Android Development Tools* (*ADT*) automatisch ein solches Archiv und signieren es mit einem speziellen Entwicklerzertifikat. Das Eclipse-Plug-in ruft hierzu Tools des Android SDK sowie des JDK auf.

Dies ist für Sie als Programmierer völlig transparent und läuft, wie Sie bereits gesehen haben, ohne weiteres Zutun ab. Die *.apk*s werden übrigens im Ordner *bin* des Projektverzeichnisses abgelegt. Sie sind aber nicht für eine Weitergabe vorgesehen.

Um ein verteilbares Archiv zu erzeugen, klicken Sie im Package Explorer die Projektwurzel mit der rechten Maustaste an und wählen Android Tools • Export Signed Application Package. Sie sehen daraufhin den Assistenten Export Android Application. Da das zu exportierende Projekt bereits ausgewählt ist, können Sie mit Next auf die Folgeseite blättern. Hier wählen Sie den Keystore aus, den Sie zum Signieren des *.apk*s verwenden möchten.

Um einen neuen zu erzeugen, klicken Sie auf Create new keystore und tragen unter Location dessen Namen und Pfad ein, zum Beispiel *C:\Users\Thomas\Entwicklung\Android-Keyring*. Vergeben Sie anschließend ein Passwort, um den Zugriff auf den Keystore zu schützen, und klicken Sie dann auf Next. Sie sehen die in Abbildung 3.6 dargestellte Seite Key Creation.

Hier müssen Sie einige Daten erfassen, die die ADT benötigen, um mithilfe des JDK-Programms *keytool* einen Keystore und Schlüssel zu erzeugen. Klicken Sie abermals auf Next. Tragen Sie nun Pfad und Namen des zu generierenden Installationsarchivs ein, und schließen Sie den Assistenten mit Finish.

Es ist sehr wichtig, Keystore und Schlüssel sicher zu verwahren und die darin enthaltenen Passwörter nicht preiszugeben. Andernfalls könnten Dritte Apps »in Ihrem Namen« verbreiten. Weitere Informationen finden Sie im Dokument *Signing Your Applications*.[5]

5 *http://developer.android.com/guide/publishing/app-signing.html*

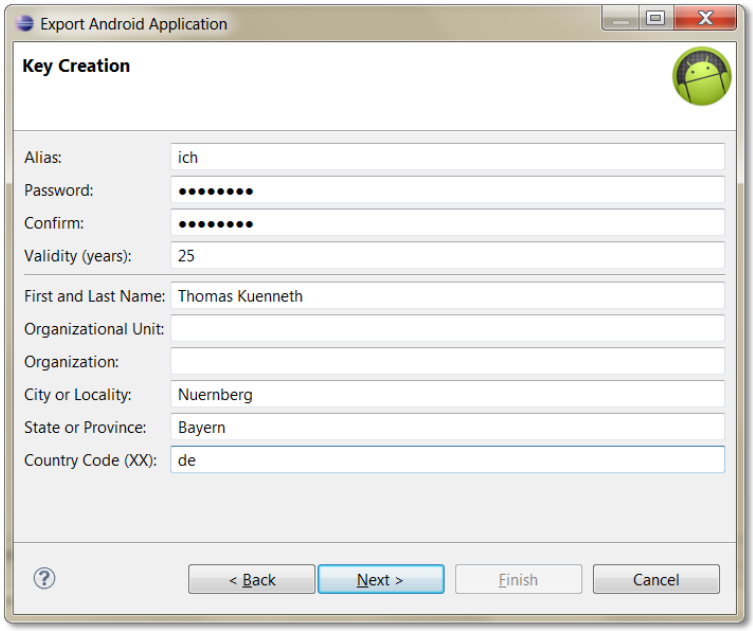


Abbildung 3.6 Dialog zum Anlegen eines Schlüssels

3.3.2 Apps in Google Play einstellen

Die von Google favorisierte Distributionsform von Apps ist die Verteilung über *Google Play*. Auf den meisten Android-Smartphones ist eine entsprechende Anwendung vorinstalliert. Sie erlaubt das Auswählen, Herunterladen und gegebenenfalls Bezahlen von Programmen. Unmittelbar nach der Anmeldung als Entwickler[6] (hierbei ist eine einmalige Gebühr von derzeit 25 US$ zu entrichten) können Sie Apps einstellen.

Der Verkauf von Programmen kann im Idealfall zu signifikanten Einnahmen führen. Leider bleiben aber auch unzählige exzellente Apps weitgehend unbeachtet. Im Hinblick auf immer wieder auftauchende Geschichten à la »Vom Tellerwäscher zum Millionär« möchte ich Ihnen deshalb ans Herz legen, mit nicht zu hohen Erwartungen ins Rennen zu gehen. Ob und wie gut sich eine Anwendung verkauft, ist von vielen Faktoren abhängig, die sich nur schwer kalkulieren lassen. Bitte prüfen Sie aber dennoch, wie Sie in steuerlicher oder abgabetechnischer Hinsicht mit solchen Einnahmen umzugehen haben.

Google unterzieht die Anwendungen erst seit Februar 2012 einer automatisierten Vorabprüfung und übernimmt sie anschließend in den Katalog. Mountain View war vorher der Meinung, dass dies unnötig ist. Jeder Entwickler, der Apps einreicht, ist als Marktteilnehmer bekannt. Beim Hochladen verpflichtet er sich, bestimmte Regeln

6 *https://play.google.com/apps/publish*

einzuhalten. Verstößt er gegen diese, kann Google als Marktbetreiber reagieren. Zudem deckt ein Bewertungs- und Kommentarsystem schwarze Schafe auf. Dennoch hat es immer wieder Malware gegeben, die sich die fehlende Vorabkontrolle zunutze gemacht hat. Googles internes Werkzeug Bouncer soll diese nun zuverlässig enttarnen (*http://googlemobile.blogspot.de/2012/02/android-and-security.html*).

Ein letzter, wichtiger Punkt: Sehr viele Funktionen (beispielsweise SMS senden, auf das Netzwerk zugreifen oder eine Nummer wählen) müssen in Form von Berechtigungen explizit angefordert werden. Diese werden während der Installation eines Programms angezeigt, so dass der Nutzer im Vorfeld schon sieht, was eine App tun möchte.

Tipp

Prüfen Sie sorgfältig, welche Berechtigungen Ihre App benötigt. Die Anwender reagieren mit harscher Kritik, wenn sich ein Programm ihrer Meinung nach zu viele oder nicht gerechtfertigte Zugriffsmöglichkeiten einräumen möchte.

Nach der Anmeldung als Entwickler können Sie mit der in Abbildung 3.7 gezeigten *Developer Console* Apps hochladen und pflegen.

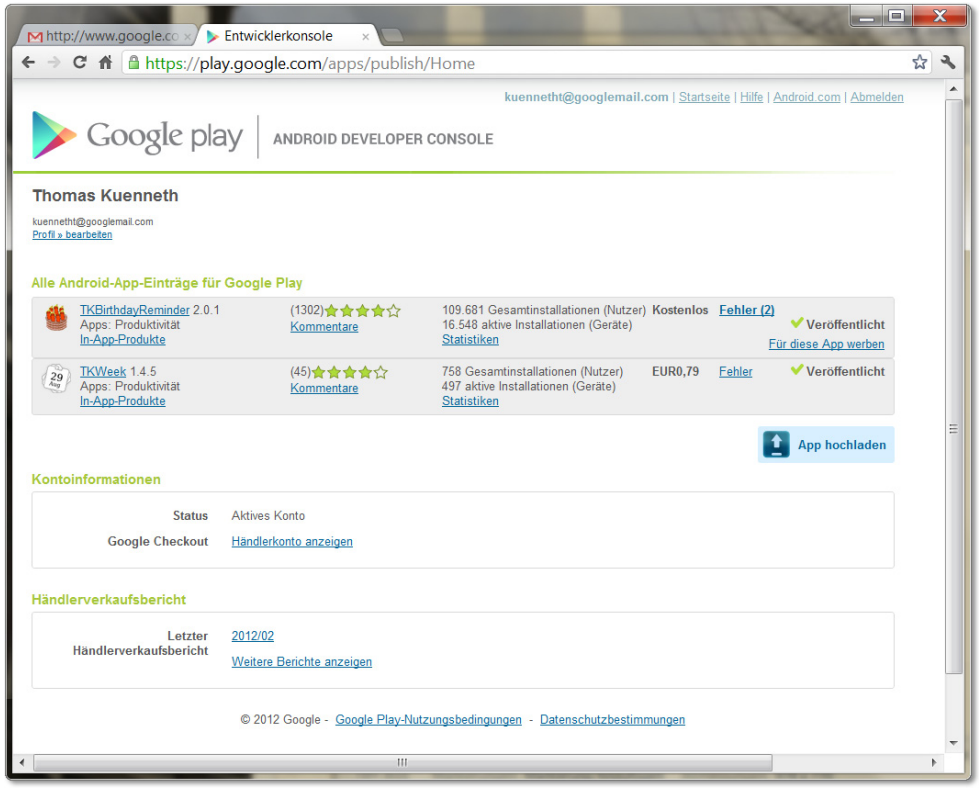


Abbildung 3.7 Die Developer Console von Google Play

Beispielsweise geben Sie (wie in Abbildung 3.8 dargestellt) die Beschreibung ein, die der Besucher beim Stöbern im Anwendungskatalog sieht. Sie können Bildschirmfotos hinterlegen, gegebenenfalls den Preis Ihres Programms ändern und Neuerungen gegenüber der letzten Version erfassen.

Warum Google Play nicht die einzige Vertriebsplattform für Ihre Software ist und wie Sie sich zusätzliche Einnahmequellen erschließen können, beschreibe ich im folgenden Abschnitt.

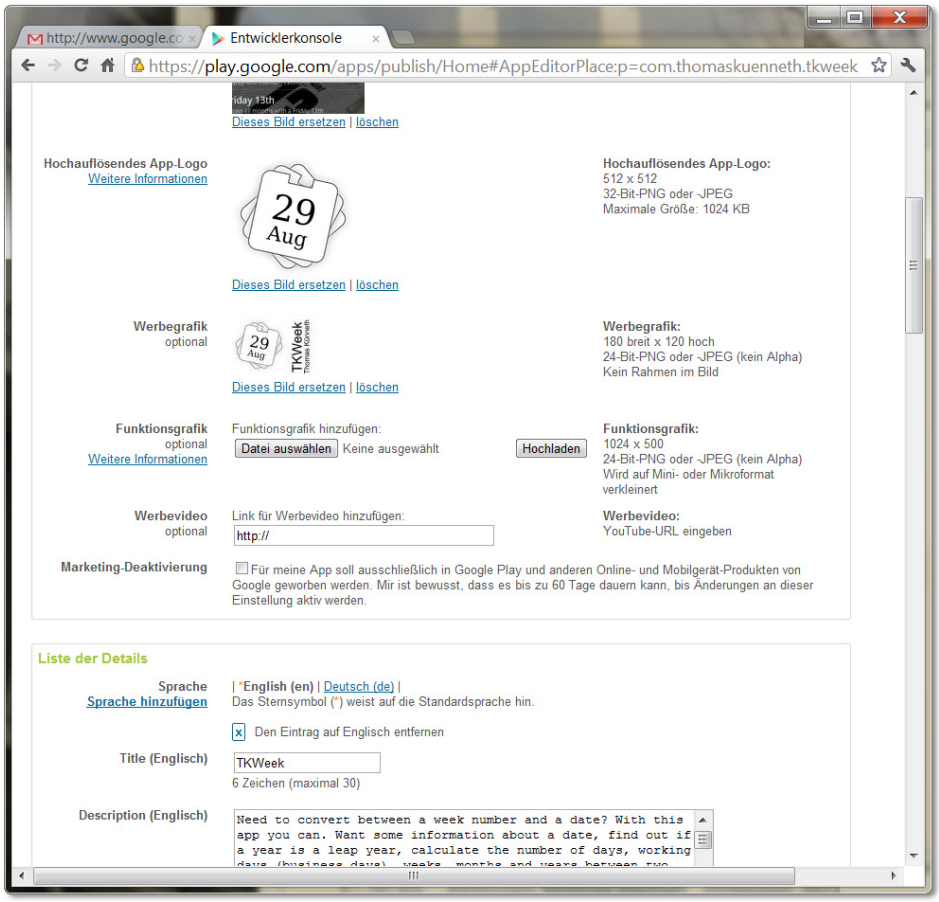


Abbildung 3.8 Produktspezifische Erfassungsseite der Developer Console

3.3.3 Alternative Märkte und Ad hoc-Verteilung

Android steht als offene Plattform jedem Gerätehersteller zur Verfügung. Allerdings sind nicht alle Komponenten Bestandteil des Open Source-Pakets. Vielleicht ist Ihnen aufgefallen, dass der Emulator die Anwendung Google Play nicht enthält. Auch Google Mail und Google Maps sind nicht auf jedem Gerät vorhanden. Die Bereitstel-

lung macht der Suchmaschinenprimus von bestimmten Eigenschaften abhängig, die beispielsweise einige der sogenannten *Portable Media Players* nicht erfüllen. Deren Anbieter haben aus der Not eine Tugend gemacht und alternative Softwareportale inklusive App für das mobile Gerät geschaffen. Auch Netzwerkbetreiber entdecken solche »Läden« zunehmend als lukrative Einnahmequelle. In diesem Zusammenhang hat sich der Begriff *Secondary Market* etabliert. Er fasst alle Angebote jenseits des Google-Pendants zusammen.

Grundsätzlich sind solche alternativen Programm-Kataloge für Sie als Entwickler als zusätzliche Vertriebsplattform für Ihr Produkt interessant. Allerdings dürfte Ihnen ab einer bestimmten Anzahl zu beliefernder Stores ein logistisches Problem drohen – mit jeder neuen Programmversion sollten Sie Ihr Angebot in jedem Appstore aktualisieren. Hier könnten neue Formen der Dienstleistung entstehen – Anbieter, die eine App automatisch in beliebige Läden provisionieren. Welche der neuen Märkte sich auf Dauer etablieren werden, ist derzeit freilich noch völlig offen.

Hinweis

Zum Zeitpunkt der Drucklegung stand Amazon kurz davor, seinen in den Vereinigten Staaten äußerst erfolgreichen *Amazon Appstore for Android* in zahlreichen weiteren Ländern zu öffnen. Ob und wie schnell Kunden ihn als Alternative zu Google Play akzeptieren, ist derzeit natürlich noch unklar.

Wenn Sie, wie weiter oben beschrieben, eine Internet-Präsenz für Ihre App aufgesetzt haben, kann die signierte Installationsdatei dort bereitgestellt und unmittelbar im Browser des Smartphones heruntergeladen werden. Das geht ganz einfach mithilfe des üblichen `<a href="">...</a>`. Allerdings müssen Nutzer auf der Einstellungsseite Anwendungen die Option Unbekannte Herkunft mit einem Häkchen versehen.

Auch mit den ADT ist die Installation fremder Anwendungen möglich. Schließen Sie hierzu das Gerät mit dem USB-Kabel an Ihren Rechner an, und öffnen Sie unter Windows die Eingabeaufforderung bzw. eine Shell unter Mac OS X oder Linux. Geben Sie anschließend `adb install <Installationsdatei>` ein. Falls `adb` nicht gefunden wird, erweitern Sie die Umgebungsvariable `PATH` um das Verzeichnis *platform-tools*.

TEIL II

Elementare Anwendungsbausteine

Kapitel 4
Activities und Broadcast Receiver

In diesem Kapitel lernen Sie zwei wichtige Bausteine von Android-Apps genauer kennen – Activities und Broadcast Receiver. Außerdem zeige ich Ihnen, wie Sie Ihre Anwendungen mit Fragmenten strukturieren.

Die Bildschirme von Smartphones sind im Vergleich zu den Anzeigen von Notebooks oder gar Desktop-Systemen winzig. Beim Bau von Apps müssen Sie sich deshalb genau überlegen, welche Informationen Sie dem Benutzer zu einem bestimmten Zeitpunkt präsentieren. Das Bewegen innerhalb des Programms, die Navigation, sollte für Ihre Anwender logisch und in sich schlüssig sein, damit diese Ihr Werk nicht frustriert zur Seite legen. Android fördert die saubere Strukturierung einer App durch Activities.

4.1 Was sind Activities?

In den ersten drei Kapiteln haben Sie schon kurz Bekanntschaft mit Activities gemacht. Diese in sich geschlossenen Komponenten beinhalten in der Regel eine Benutzeroberfläche. Sie repräsentieren Aktionen wie *Anruf tätigen*, *eine SMS senden*, *Termindetails bearbeiten* oder *einen Monatskalender anzeigen*. Jeder Activity steht ein Fenster für ihre Bedienelemente zur Verfügung. Normalerweise füllt dies den Bildschirm aus. Es kann aber auch kleiner sein und über anderen Fenstern schweben.

4.1.1 Struktur von Apps

Eine App besteht meist aus mehreren Activities. In welcher Reihenfolge diese aufgerufen werden, kann von Aktionen des Benutzers abhängen (beispielsweise, indem er einen Befehl in der Action Bar antippt) oder durch die Programmlogik vorgegeben sein. Wird in einer Activity eine Liste von Kontakten angezeigt, führt das Antippen eines Listenelements zur Detailansicht. Hier navigiert der Anwender nicht bewusst zu einer anderen »Seite« – das Programm tut dies für ihn.

Jede App sollte eine Hauptactivity haben, die beim ersten Programmstart aufgerufen wird. Oftmals handelt es sich hierbei um eine Art Auswahl- oder Menüseite, die zu den eigentlichen »Modulen« verzweigt. Wie diese definiert wird, ist im folgenden Abschnitt beschrieben.

Hinweis

Tablets haben im Vergleich zu Smartphones große Bildschirme. Auf ihnen können Auswahllisten und Detailansichten problemlos gleichzeitig dargestellt werden. Wie Sie später noch sehen werden, kennt Android sogenannte größen- oder auflösungsabhängige Ressourcen, mit denen sich für jede Bildschirmauflösung und -größe die optimale Benutzeroberfläche vorhalten lässt.

Der Fluss innerhalb einer App und über Anwendungsgrenzen hinweg entsteht also (unter anderem) durch das Starten von Folgeaktivitäten. Die zu diesem Zeitpunkt ausgeführte Activity wird vom System angehalten und auf einen Stapel, den sogenannten *Back Stack*, gelegt. Drückt der Benutzer die ZURÜCK-Taste (mit der Einführung von Android 3 wurde diese »virtualisiert« und erscheint als Element der *System Bar*), oder wird die nun aktive Activity auf eine andere Weise beendet, entfernt das System sie von diesem Stapel und reaktiviert die »darunterliegende«. Ausführliche Informationen zum Lebenszyklus von Activities finden Sie in Abschnitt 4.1.2, »Lebenszyklus von Activities«.

Die Manifestdatei

Wie Sie bereits wissen, gehört zu jeder Android-App eine zentrale Beschreibungsdatei. Sie enthält eine Liste der Komponenten, aus denen das Programm besteht. Außerdem werden in ihr die benötigten Berechtigungen sowie etwaige zusätzlich verwendete Bibliotheken vermerkt. Auch Angaben zur mindestens nötigen oder gewünschten Android-Version tragen Sie hier ein. Die Datei *AndroidManifest.xml* befindet sich auf der obersten Ebene des Projektverzeichnisses. Sie wird durch den Assistenten zum Anlegen neuer Projekte generiert.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.thomaskuenneth"
          android:versionCode="1"
          android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        <activity android:name=".TestActivity"
                  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 4.1 AndroidManifest.xml

Das Attribut `package` enthält seinen Wert aus dem Feld PACKAGE NAME des Assistenten. Die bei MINIMUM SDK eingetragene Zahl findet sich in der Manifestdatei im Attribut `android:minSdkVersion` des Elements `<uses-sdk />`. `android:versionCode` und `android:versionName` geben die Versionsnummer der Anwendung an.

Während `versionCode` eine für die programmatische Auswertung gedachte Zahl ist, enthält `versionName` die Versionsnummer in einer für den Anwender verständlichen Form, zum Beispiel 1.2 oder 1.2.3. Google schlägt vor, für die erste veröffentlichte Version einer App `android:versionCode` auf 1 zu setzen und mit jedem Update beispielsweise um 1 zu erhöhen. Sie sollten stets beide Werte angeben.

Die Komponenten einer Anwendung sind Kinder des Elements `<application />`. Wenn Sie im Assistenten zum Anlegen neuer Projekte CREATE ACTIVITY mit einem Häkchen versehen und einen Namen eintragen, enthält das Manifest ein Element `<activity />`. Dessen Attribut `android:name` beinhaltet den im Assistenten eingegebenen Activity-Namen. Haben Sie den führenden Punkt vor `TestActivity` bemerkt? Er ist ein Hinweis auf einen »fehlenden« Paketteil. In so einem Fall wird der Inhalt des Attributs `package` eingefügt.

Die genaue Bedeutung des Elements `<intent-filter />` erkläre ich Ihnen etwas später. Fürs Erste soll der Hinweis genügen, dass Sie auf diese Weise eine Activity zur Hauptaktivität machen. Sie lässt sich über den Programmstarter oder eine Verknüpfung auf dem Home Screen aufrufen.

Trennung von Programmlogik und Ressourcen

Sowohl das Element `<application />` als auch `<activity />` enthalten das Attribut `android:label`. Dessen Wert ist in beiden Fällen der gleiche, nämlich die Zeichenkette `@string/app_name`. Wie Sie bereits aus Abschnitt 2.2, »Die Benutzeroberfläche«, wissen, enthält die Datei *strings.xml* im Verzeichnis *res/values* Schlüssel-Wert-Paare, die eine Zeichenkette einem Bezeichner zuordnen. Der Wert des Attributs `android:label` ergibt sich also aus dem Schlüssel `app_name`, der in *strings.xml* eingetragen wurde. Ein Blick in diese Datei zeigt, dass beim Anlegen des Projekts der Name der App übernommen wurde.

Die Speicherung von Texten an einem zentralen Ort hat zahlreiche Vorteile. Beispielsweise werden identische Textteile leichter entdeckt, als wenn diese in den

Quelltexten der Klassen verborgen sind. Damit lässt sich – wenn auch in eher bescheidenem Umfang – Speicherplatz sparen. Außerdem macht die Trennung von Daten und Programmlogik die Internationalisierung, also die Übersetzung einer App in verschiedene Sprachen, viel einfacher.

Hierzu wird für jede zu unterstützende Sprache im Ordner *res* ein Verzeichnis angelegt. Dessen Name beginnt mit *values-* und endet mit dem ISO-Sprachschlüssel. Für Deutsch ist dies *de*. Das Verzeichnis muss also *values-de* heißen. Jeder dieser Ordner erhält eine eigene Version von *strings.xml*. Deren Bezeichner sind stets gleich, die Texte liegen hingegen in den jeweiligen Sprachen vor. Texte in der Standardsprache verbleiben in *values*. Im folgenden Beispiel wird Deutsch als Standardsprache verwendet:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   ...
   <!-- Willkommensmeldung -->
   <string name="willkommen">
      Guten Tag. Schön, dass Sie mich gestartet haben.
      Bitte verraten Sie mir Ihren Namen.
   </string>
   ...
```

Listing 4.2 Auszug aus res/values/strings.xml

Wird die App auf einem Gerät ausgeführt, dessen Sprache Englisch ist, wird auf diese Datei zugegriffen:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   ...
   <!-- Willkommensmeldung -->
   <string name="willkommen">
      Thank you very much for launching this app.
      Please enter your name.
   </string>
   ...
```

Listing 4.3 Auszug aus res/values-en/strings.xml

Bei der Auflösung einer Referenz wie `@string/app_name` geht Android folgendermaßen vor: Wenn für die aktuell eingestellte Sprache eine Datei *strings.xml* existiert und diese den benötigten Bezeichner enthält, wird der ihm zugeordnete Text verwendet. Andernfalls wird auf *strings.xml* in *values* zugegriffen. Konkret bedeutet dies: Alle referenzierten Bezeichner müssen sich für die Standardsprache auflösen

lassen. Gelingt dies nicht, läuft die App zur Laufzeit auf einen Fehler. Lokalisierungen, also Übersetzungen in andere Sprachen, müssen hingegen nicht vollständig vorliegen.

Wenn Sie eine App über *Google Play* vertreiben möchten, sollten Sie Englisch als Standardsprache verwenden. Auf diese Weise maximieren Sie die Zahl potenzieller Nutzer. Tragen Sie hierzu in *values/strings.xml* stets englische Texte ein, und fügen Sie in *values-de/strings.xml* entsprechende deutsche Übersetzungen hinzu.

Tipp

Mit der App *Custom Locale*, die in jedem Emulator-Systemabbild zur Verfügung steht, können Sie bequem testen, ob Ihr Programm stets die erwarteten Texte ausgibt.

Auf sehr ähnliche Weise lässt sich der Wert des Attributs `android:icon` auflösen. Der Ordner *res* enthält einige Unterverzeichnisse, die mit *drawable* beginnen. *.png*- und *.jpg*-Grafiken, die dort hineinkopiert werden, können über die Zeichenkette `@drawable/<Dateiname ohne Erweiterung>` referenziert werden. Die Datei *icon.png* ist also mithilfe des Ausdrucks `@drawable/icon` erreichbar. Wenn Sie die Kopien in *drawable-hdpi* und *drawable-ldpi* vergleichen, stellen Sie fest, dass sich die Dateien in Größe und Auflösung unterscheiden.

Vielleicht fragen Sie sich nun, warum Bilder in unterschiedlichen Verzeichnissen über einen gemeinsamen Bezeichner angesprochen werden sollten. Android-Geräte können sich in vielerlei Hinsicht unterscheiden. Neben Smartphones mit beispielsweise 3,2- oder 4-Zoll-Bildschirm gibt es Tablets, deren Bildschirmdiagonalen bis zu 10 Zoll betragen. Auch die Zahl der horizontal und vertikal darstellbaren Pixel variiert. Um den durch diese Vielfalt entstehenden Aufwand für Entwickler in Grenzen zu halten, definiert Android einige Klassen für Bildschirmgrößen und Pixeldichten.

Fordert eine App zur Laufzeit eine Grafik an, sucht das System die am besten zur Hardware passende aus. Die Größe von Programm-Icons hat Google im Dokument *Icon Design Guidelines* definiert:[1] 36 × 36 Pixel für Geräte mit niedriger Pixeldichte (ldpi), 48 × 48 Pixel für mittlere Dichte (mdpi), 72 × 72 Pixel für hohe Dichte (hdpi) und 96 × 96 Pixel für sehr hohe Dichte (xhdpi). Die Varianten von *icon.png*, die beim Anlegen eines Projekts automatisch in die korrespondierenden *drawable-...*-Verzeichnisse kopiert werden, entsprechen natürlich diesen Vorgaben.

Android stellt zahlreiche Methoden zur Verfügung, um auf Ressourcen zuzugreifen. Die folgende Activity zeigt einige Beispiele. Sie finden den Quelltext der Klasse `ZugriffAufRessourcenActivity` im Verzeichnis *Quelltexte/Diverses* der Begleit-DVD

1 *http://developer.android.com/guide/practices/ui_guidelines/icon_design.html*

des Buches. Um sie auszuprobieren, legen Sie am besten ein neues Android-Projekt an und kopieren anschließend die Datei. Sie gibt drei Zeilen in die Sicht LogCat aus: den Namen Ihrer App, den aktuellen Tag des Jahres und das Jahr sowie zwei `boolean`-Werte:

```
package com.thomaskuenneth;

import java.util.Calendar;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class ZugriffAufRessourcenActivity extends Activity {

   private static final String TAG =
      ZugriffAufRessourcenActivity.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Log.d(TAG, "getString(R.string.app_name): "
            + getString(R.string.app_name));
      Calendar cal = Calendar.getInstance();
      Log.d(TAG,
            getString(R.string.datum,
            "Heute ist der",
            cal.get(Calendar.DAY_OF_YEAR),
            cal.get(Calendar.YEAR)));
      boolean b1 = getResources().getBoolean(R.bool.bool1);
      boolean b2 = getResources().getBoolean(R.bool.bool2);
      Log.d(TAG, "b1=" + b1 + ", b2=" + b2);
   }
}
```

Listing 4.4 ZugriffAufRessourcenActivity.java

Damit Ihre App funktioniert, müssen Sie die Ressourcen Ihres Programms erweitern. Fügen Sie in *strings.xml* diese Zeile ein:

```
<string name="datum">%1$s %2$d. Tag des Jahres %3$d</string>
```

Zur Laufzeit der App wird daraus zum Beispiel der Text »Heute ist der 42. Tag des Jahres 2011«. Ich habe einen Teil des auszugebenden Strings im Quelltext belassen, um

Ihnen die Nachvollziehbarkeit zu erleichtern. In Ihrer App würden Sie auch den Text »Heute ist der« auslagern.

Dem Bezeichner `datum` wird ein Text mit drei Platzhaltern zugewiesen. Die Ziffern 1, 2 und 3 jeweils hinter einem Prozentzeichen geben die Reihenfolge an, in der sie beim Aufruf der Methode `getString(R.string.datum, ...)` mit Inhalten gefüllt werden. Die Buchstaben `s` und `d` jeweils hinter einem Dollarzeichen kennzeichnen den Typ; `s` steht für String und `d` für Dezimalzahl.

Auf diese Weise können Sie Texte mit Werten anreichern, die erst zur Laufzeit bekannt sind. Übrigens müssen Sie bei der Übersetzung in andere Sprachen nicht auf die Reihenfolge der Platzhalter achten, weil Sie durch die Angabe der Ziffer ja festlegen, auf welchen Parameter in `getString()` Sie sich beziehen. Das ist nicht nur bei Monats- und Jahresangaben sehr praktisch.

Die dritte Ausgabeanweisung in der Klasse schreibt zwei `boolean`-Werte in die Sicht LogCat. Diese werden mit dem Ausdruck `getResources().getBoolean()` ermittelt. Aber woher kommen die beiden Bezeichner `bool1` und `bool2`? Erzeugen Sie im Ordner *res/values* die Datei *diverses.xml*, und fügen Sie die folgenden Zeilen ein:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="bool1">true</bool>
    <bool name="bool2">false</bool>
</resources>
```

Listing 4.5 diverses.xml

Sogenannte *einfache Ressourcen* werden über ihr `name`-Attribut referenziert. Neben `boolean`-Werten können Sie unter anderem Farben und Integerzahlen in einer gemeinsamen XML-Datei ablegen. Deren Name ist, wie Sie gesehen haben, frei wählbar. Weiterführende Informationen finden Sie im Dokument *More Resource Types*.[2]

Wie Sie bereits aus früheren Kapiteln wissen, spiegeln Activities für den Anwender die Funktionen Ihrer App wieder. Ob ein Programm einfach zu bedienen ist, hängt gerade auf Smartphones nicht nur von der Ausgestaltung der Bedienoberfläche ab, sondern auch von seiner Navigierbarkeit: Wie schnell und intuitiv gelingt dem Benutzer, was er mit der App erledigen wollte? `Activity` und die von ihr abgeleiteten Klassen versuchen, Sie als Entwickler beim Bau von einfach zu bedienenden Apps zu unterstützen. Im folgenden Abschnitt stelle ich Ihnen deshalb die Struktur und den Lebenszyklus von Activities ausführlicher vor.

2 *http://developer.android.com/guide/topics/resources/more-resources.html*

4.1.2 Lebenszyklus von Activities

Activities werden sowohl vom System als auch von anderen Activities aufgerufen. Damit dies funktioniert, müssen Sie die Activities Ihrer App in der Manifestdatei eintragen. Jede Aktivität erhält ein eigenes `<activity />`-Element. Dessen Attribut `android:name` verweist auf den in der Regel voll qualifizierten Klassennamen. Es muss zwingend vorhanden sein. Fehlt der Paketteil des Klassennamens, beginnt der Eintrag also mit einem Punkt, wird das im Attribut `package` eingetragene Paket substituiert.

Die Klasse Activity

Jede App sollte eine Hauptaktivität definieren, die beispielsweise beim Antippen des Programm-Icons im Anwendungsstarter aufgerufen wird. In der Manifestdatei fügen Sie ihr deshalb ein `<intent-filter />`-Element hinzu. Dessen Kindelement `<action />` kennzeichnet die Activity als Haupteinstiegspunkt in die Anwendung. `<category />` sorgt dafür, dass sie im Programmstarter angezeigt wird.

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

Der Assistent zum Anlegen von Projekten tut dies automatisch, wenn Sie Create Activity mit einem Häkchen versehen und einen Namen für die anzulegende Aktivität eintragen. Die auf diese Weise generierte Klasse ist sehr einfach gehalten, zeigt aber einige grundlegende Vorgehensweisen beim Bau von Activities:

```
package com.thomaskuenneth;

import android.app.Activity;
import android.os.Bundle;

public class Test extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Listing 4.6 Test.java

Activities leiten von `android.app.Activity` oder deren Kindern ab. Vor allem die Klasse `android.app.ListActivity` wird gerne verwendet. Sie stellt beliebige Elemente

in Listenform dar. Beispiele für ihren Einsatz sind Posteingänge oder Nachrichtenübersichten, Kontaktlisten und Twitter-Timelines. Das Antippen einer Zeile führt normalerweise zu einer Detailansicht, die den ausgewählten Eintrag vollständig anzeigt.

Mit der ebenfalls häufig eingesetzten Klasse `PreferenceActivity` realisieren Sie Programmeinstellungsseiten. Im Gegensatz zur `ListActivity` wird sie eher selten als Hauptaktivität verwendet.

Die Methode `onCreate()` wird von praktisch jeder selbst geschriebenen Activity überschrieben. Android ruft sie während der Initialisierungsphase einer Aktivität auf. Sie erledigt normalerweise folgende Aufgaben:

1. Aufrufen der gleichnamigen Elternmethode
2. Initialisieren von Instanzvariablen
3. Setzen der Benutzeroberfläche
4. Wiederherstellen eines gespeicherten Zustands

Der Aufruf von `super.onCreate()` ist obligatorisch. Unterbleibt er, wird zur Laufzeit eine `SuperNotCalledException` geworfen. Das Setzen der Benutzeroberfläche erfolgt typischerweise durch die Anweisung `setContentView()`. Der übergebene Parameter, zum Beispiel `R.layout.main`, referenziert ein sogenanntes Layout. Es wird in einer gleichnamigen XML-Datei (beispielsweise *main.xml*) gespeichert. Ausführliche Hinweise zum Bau der Benutzeroberfläche finden Sie in Kapitel 5, »Benutzeroberflächen«.

Einige Kindklassen von `Activity` setzen von sich aus ein Layout. In solchen Fällen dürfen Sie `setContentView()` natürlich nicht aufrufen. Dies ist zum Beispiel bei `ListActivity` der Fall. Wie Sie solche Activities mit Daten füllen, entnehmen Sie bitte der jeweiligen Seite in Googles Entwicklerdokumentation. Ein Beispiel finden Sie in Kapitel 3, »Von der Idee zur Veröffentlichung«.

Das Wiederherstellen eines gespeicherten Zustands ist praktisch, um »Wiederanlaufzeiten« einer Activity zu optimieren. Wie Sie im folgenden Abschnitt noch ausführlicher sehen werden, startet und stoppt Android Activities unter bestimmten Umständen automatisch. Ein Grund ist der Orientierungswechsel, also das Drehen des Geräts vom Hochkant- in das Querformat (oder umgekehrt). In so einem Fall beendet Android die laufende Activity und startet sie im Anschluss daran wieder. Was sich auf den ersten Blick vielleicht absurd anhört, ist durchweg praktisch: Oft möchte man die Benutzeroberfläche in Abhängigkeit von der Haltung des Geräts anordnen.

Um den zweiten Start zu beschleunigen, kann man beim Beenden einer Activity Daten in einem Zwischenspeicher (den *instance state*) ablegen. Dieser wird bei einem erneuten Start übergeben. Wie das funktioniert, demonstriert die Activity `InstanceStateDemo`. Sie finden sie auf der Begleit-DVD des Buches im Verzeichnis *Quelltexte/Diverses*.

Der an `onCreate()` übergebene Parameter `savedInstanceState` verweist auf den weiter oben angesprochenen Zwischenspeicher. Er hat den Typ `android.os. Bundle` und sammelt Schlüssel-Wert-Paare. Als Schlüssel werden Strings verwendet. Werte können unter anderem primitive Datentypen sowie Felder von diesen sein, aber auch solche, die das Interface `android.os.Parcelable` implementieren.

Wenn Sie die App starten, wird zunächst nur der Text *savedInstanceState war null* in die Sicht LogCat geschrieben. Ändern Sie nun die Orientierung des Emulators, indem Sie die Tasten [7] oder [9] auf dem Ziffernblock Ihrer Tastatur drücken. Android beendet daraufhin die Activity und startet sie neu. Da beim zweiten Start die `Bundle`-Referenz nicht mehr `null` ist, ändert sich die Konsolenausgabe entsprechend.

```
package com.thomaskuenneth;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class InstanceStateDemo extends Activity {

   private static final String TAG =
      InstanceStateDemo.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     if (savedInstanceState == null) {
        Log.d(TAG, "savedInstanceState war null");
     } else {
        Log.d(TAG,."wurde vor "
              + (System.currentTimeMillis() - savedInstanceState
                        getLong(TAG)) +
                        " Millisekunden beendet");
     }
   }

   @Override
   protected void onSaveInstanceState(Bundle outState) {
      super.onSaveInstanceState(outState);
      outState.putLong(TAG, System.currentTimeMillis());
   }
}
```

Listing 4.7 InstanceStateDemo.java

Um beim Beenden Ihrer Activity Daten in einem Bundle abzulegen, überschreiben Sie einfach die Methode `onSaveInstanceState()` und nutzen die `put...()`-Methoden der übergebenen `Bundle`-Referenz. Innerhalb von `onCreate()` verwenden Sie korrespondierende `get...()`-Aufrufe, um Daten wieder auszulesen. Beachten Sie hierbei aber, dass die Referenz `null` sein kann. Um `NullPointerException`s zu vermeiden, müssen Sie Zugriffe auf das Bundle in jedem Fall mit einer entsprechenden `if`-Abfrage versehen.

Vielleicht fragen Sie sich nun, welche Daten Sie für einen erneuten Wiederanlauf sichern sollten. Inhalte oder Status von Bedienelementen, beispielsweise Eingaben in Textfelder, können automatisch durch das System wiederhergestellt werden und müssen deshalb nicht gespeichert werden. Damit das klappt, müssen Sie beim Überschreiben von `onSaveInstanceState()` als Erstes die Elternmethode aufrufen.

Sehr praktisch ist der hier vorgestellte Mechanismus für Werte, deren Berechnung oder Ermittlung zeitaufwendig ist. Dies ist unter anderem bei Webservice-Aufrufen der Fall. Allerdings – und diese Einschränkung ist sehr wichtig – dürfen Sie mit diesem Mechanismus nur transiente Daten ablegen. Android garantiert nämlich nicht, dass `onSaveInstanceState()` immer aufgerufen wird. Der richtige Ort für das Persistieren, beispielsweise das Schreiben in eine Datenbank, ist hingegen die Methode `onPause()`. Diese lernen Sie im Folgenden kennen.

Tipp

Sie können das Wiederherstellen eines früheren Zustands auch aus `onCreate()` auslagern, indem Sie stattdessen die Methode `onRestoreInstanceState()` überschreiben. Diese wird nach der Abarbeitung von `onStart()` aufgerufen.

Wichtige Callback-Methoden

Mit `onCreate()` und `onSaveInstanceState()` haben Sie bereits zwei sehr wichtige Methoden einer Activity kennengelernt. In der Klasse `ActivityLifecycleDemo` habe ich einige weitere Methoden überschrieben. Um zu verstehen, wann diese aufgerufen werden, sollten Sie eine App erstellen, die diese Activity beinhaltet. Sie finden das vollständige Projekt *ActivityLifecycleDemo* im Verzeichnis *Quelltexte* der Begleit-DVD.

```
package com.thomaskuenneth.activitylifecycledemo;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class ActivityLifecycleDemo extends Activity {

   private static final String TAG =
      ActivityLifecycleDemo.class.getSimpleName();

   private static int zaehler = 1;

   private int lokalerZaehler = zaehler++;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      log("onCreate");
      setContentView(R.layout.main);
      TextView tv = (TextView) findViewById(R.id.textview);
      tv.setText(getString(R.string.msg, lokalerZaehler));

      Button buttonNew = (Button) findViewById(R.id.id_new);
      buttonNew.setOnClickListener(new OnClickListener() {

         @Override
         public void onClick(View v) {
            Intent i = new Intent(ActivityLifecycleDemo.this,
                  ActivityLifecycleDemo.class);
            startActivity(i);
         }
      });

      Button buttonFinish =
         (Button) findViewById(R.id.id_finish);
      buttonFinish.setOnClickListener(new OnClickListener() {

         @Override
         public void onClick(View v) {
            finish();
         }
      });
   }
```

```
    @Override
    protected void onStart() {
        super.onStart();
        log("onStart");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        log("onRestart");
    }

    @Override
    protected void onResume() {
        super.onResume();
        log("onResume");
    }

    @Override
    protected void onPause() {
        super.onPause();
        log("onPause");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        log("onDestroy");
    }

    private void log(String methodName) {
        Log.d(TAG, methodName + "() #" + lokalerZaehler);
    }
}
```

Listing 4.8 ActivityLifecycleDemo.java

Die in Abbildung 4.1 dargestellte Beispiel-App *ActivityLifecycleDemo* hat eine sehr einfache Benutzeroberfläche, zwei Schaltflächen und ein Textfeld. Dieses zeigt nach dem ersten Start die Meldung »Ich habe die laufende Nummer 1« an. Zu diesem Zeitpunkt wurden die Methoden `onCreate()`, `onStart()` und `onResume()` (in dieser Reihenfolge) abgearbeitet.

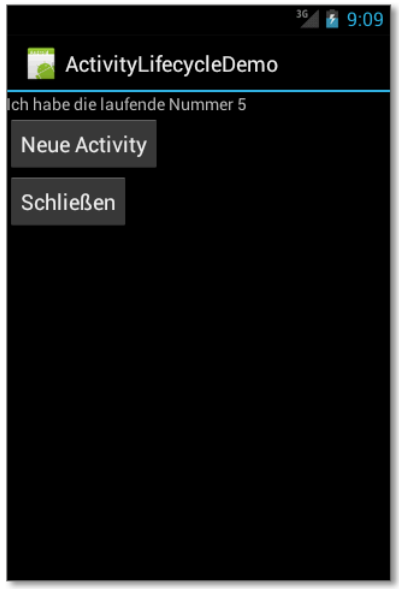


Abbildung 4.1 Die Beispielanwendung ActivityLifecycleDemo

Klicken Sie nun auf NEUE ACTIVITY. Nach einer kurzen Animation erscheint die Activity *ActivityLifecycleDemo* erneut. Dieses Mal wird die Meldung »Ich habe die laufende Nummer 2« ausgegeben. Ein Blick in die Sicht LOGCAT zeigt, dass die Methode `onPause()` vor den drei Methoden `onCreate()`, `onStart()` und `onResume()` der »neuen« Aktivität durchlaufen wurde.

Drücken Sie nun die ZURÜCK-Taste des Emulators. Wie erwartet ist wieder die erste Activity zu sehen. Neben deren Methoden `onRestart()`, `onStart()` und `onResume()` hat Android zusätzlich `onPause()` und `onDestroy()` der beendeten Aktivität durchlaufen. Wie Sie bereits wissen, verwaltet Android Activities auf einem Stapel, dem sogenannten *Back Stack*. Die am weitesten oben liegende Aktivität ist aktiv, wird also vom Benutzer bedient.

Durch Drücken der ZURÜCK-Taste bzw. durch Anklicken des virtuellen Pendants in der System Bar wird sie vom Stapel entfernt, und die Activity, die zuletzt vor ihr auf den Back Stack gelegt wurde, läuft wieder an. Der Stapel arbeitet also nach dem klassischen Prinzip »last in, first out«. Ausführliche Informationen hierzu finden Sie im Dokument *Tasks and Back Stack*.[3]

Sie können eine Activity programmatisch beenden, indem Sie deren Methode `finish()` aufrufen. Die Entwicklerdokumentation rät zwar dazu, dies nach Möglich-

3 *http://developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html*

keit nicht zu tun, da der Anwender nämlich erwartet, explizit ZURÜCK drücken oder anklicken zu müssen, um zur vorhergehenden Aktivität zurückzukehren.

Allerdings gibt es gute Gründe, die Methode in bestimmten Situationen zu nutzen. Denken Sie an Activities, in denen der Benutzer neue Daten eingibt oder bestehende verändert. In diesem Fall müssen Sie dem Anwender sogar die Wahl lassen, Änderungen zu übernehmen oder zu verwerfen. Dies wird üblicherweise mit Schaltflächen realisiert.

Im Grunde kennen Activities drei Zustände:

- Die Activity ist im Vordergrund und wird durch den Anwender bedient. Google nennt diesen Status *resumed* oder *running*.
- Die Activity befindet sich nicht mehr im Vordergrund und hat nicht den Fokus, ist aber noch sichtbar. Dies ist der Fall, wenn die laufende Activity nicht den gesamten Bildschirm ausfüllt oder zum Teil transparent ist. Dieser Zustand wird *paused* genannt.
- Die Activity befindet sich im Hintergrund. Sie wird vollständig von einer anderen bedeckt und gilt deshalb als *stopped*.

Pausierende und gestoppte Aktivitäten verbleiben vollständig im Speicher, behalten also den Zustand ihrer Klassen- und Instanzvariablen. Eine Activity im Zustand *paused* ist weiterhin an den Fenstermanager angebunden – sie ist zum Teil sichtbar. Deshalb wird sie vom System nur in Notsituationen zerstört, beispielsweise wenn der Arbeitsspeicher sehr knapp wird. Activities mit Status *stopped* hingegen wurden bereits vom Fenstermanager abgekoppelt. Ihr Speicher wird deshalb freigegeben, wenn er anderweitig benötigt wird.

Tipp

Nutzen Sie die Möglichkeit, die Methode `onPause()` zu überschreiben, um wichtige Daten Ihrer App zu sichern.

4.2 Kommunikation zwischen Anwendungsbausteinen

Aus der Sicht des Benutzers repräsentieren Activities die Funktionen einer App. Technisch gesehen, handelt es sich bei diesen Grundbausteinen um klassische Komponenten. Denn Activities haben einen Zustand und können, wie Sie in diesem Abschnitt sehen werden, Nachrichten senden und empfangen.

4.2.1 Intents

Nachrichten werden in sogenannten *Intents* gekapselt. Android nutzt diese Boten aber nicht nur für die lose Kopplung von Activities, sondern auch für den Datenaustausch zwischen den anderen elementaren Anwendungsbausteinen *Service* und *Broadcast Receiver*.

Intents binden die Komponenten einer oder mehrerer Apps zur Laufzeit. Sie sind passive Datenstrukturen, die entweder die abstrakte Beschreibung einer auszuführenden Operation enthalten oder über ein eingetretenes Ereignis informieren. Auf welche Weise sie zugestellt werden, hängt von der Art der Komponente ab. Entsprechende Beschreibungen finden Sie deshalb in den Abschnitten zu den jeweiligen Komponenten.

Aufbau von Intents

Intent-Objekte übermitteln ihren Empfängern, welche Aktionen diese ausführen sollen, und liefern die hierfür benötigten Daten zum Teil gleich mit. Die sogenannte Kategorie hingegen wird durch Android selbst ausgewertet. Sie beschreibt, welche Komponenten ein Intent behandeln bzw. erhalten kann und wie diese gegebenenfalls zu starten sind.

Sie können den »Empfänger« eines Intents unmittelbar angeben, indem Sie den *Komponentennamen* setzen. Er besteht aus dem voll qualifizierten Klassennamen der Zielkomponente (eine Activity, ein Service oder ein Broadcast Receiver) und dem Paketnamen der App, die diese Komponente enthält. Sie finden diesen zum Beispiel im Attribut `package` der Manifestdatei. Beachten Sie, dass der Paketname der Anwendung und das Paket der Komponente keineswegs identisch sein müssen.

Intents, die die Zielkomponente mit einem Komponentennamen benennen, werden *explizite Intents* genannt. Da Paket- und Klasseninformationen Entwicklern fremder Apps normalerweise nicht bekannt sind, werden solche Intents üblicherweise für den internen Nachrichtenaustausch verwendet. *Implizite Intents* hingegen nennen kein Ziel – der Komponentenname bleibt leer. Sie werden üblicherweise verwendet, um Komponenten anderer Apps zu aktivieren.

Vielleicht fragen Sie sich, wie der Empfänger ermittelt wird. Das System muss ermitteln, welche Komponenten am besten für die Behandlung bzw. Auswertung des Intents geeignet sind. Das können einzelne Activities oder Services sein, die die gewünschte Aktion ausführen. Die bestmöglichen Empfänger können aber auch Broadcast Receiver sein. Diese reagieren auf das Ereignis, das mit dem Intent beschrieben wurde. Hierzu wird der Inhalt eines Intents mit sogenannten *Intent-Filtern* verglichen.

Inhalt von Intents und Intent-Filtern

Intent-Filter beschreiben, auf welche impliziten Intents eine Komponente reagieren möchte bzw. welche sie empfangen kann. Sie werden in der Manifestdatei eingetragen. Hierzu wird jedes Wurzelelement einer Komponente (beispielsweise `<activity />`) mit dem Kindelement `<intent-filter />` versehen. Werden keine Intent-Filter angegeben, kann die Komponente nur explizite Intents empfangen. Implizite Intents werden zugestellt, sofern ihre Eigenschaften zu den durch die Kinder des Elements `<intent-filter />` angegebenen Filterkriterien passen.

Das Element `<action />` muss mindestens einmal vorhanden sein. Es speichert die Aktion, die ein empfangenes Intent auslösen soll. Beispiele hierfür sind *Anruf tätigen*, *Fehlerbericht senden*, *Suche im Web durchführen* und *Verknüpfung anlegen*. Die Klasse `Intent` enthält Konstanten für diese und viele weitere Aktionen. Deren Namen beginnen mit dem Präfix `ACTION_`.

Um eine dieser Actions in der Manifestdatei anzugeben (weil Ihre Activity, Ihr Service oder Broadcast Receiver darauf reagieren soll), wird dem Attribut `android:name` des Elements `<action />` die Zeichenkette `android.intent.action.`, gefolgt vom Konstantennamen der Action ohne das Präfix `ACTION_`, zugewiesen.

Hierzu zwei Beispiele: Möchten Sie, dass eine Ihrer Komponenten auf die Action `ACTION_MAIN` reagiert, weisen Sie `android:name` den Wert `android.intent.action.MAIN` zu. `ACTION_WEB_SEARCH` wird zu `android.intent.action.WEB_SEARCH`. Wie Sie Intents, die solche Aktionen auslösen, an Activities oder Broadcast Receiver übermitteln, zeige ich Ihnen in den folgenden Abschnitten. Ausführliche Informationen zu Services finden Sie in Kapitel 6, »Multitasking«.

Die Kategorien eines Intents legen fest, an welche Arten von Komponenten es übermittelt werden kann. Die Klasse `Intent` enthält Konstanten, die einige vordefinierte Kategorien repräsentieren. Beispielsweise bedeutet `CATEGORY_BROWSABLE`, dass eine Aktivität vom Webbrowser aufgerufen werden kann, um bestimmte Inhalte anzuzeigen.

Wenn eine Ihrer Activities diese Fähigkeit besitzt und Sie diese anderen Apps zur Verfügung stellen möchten, erweitern Sie das Element `<intent-filter />` in der Manifestdatei um `<category />` und weisen dessen Attribut `android:name` den Wert `android.intent.category.BROWSABLE` zu. Auch hier wird einem Präfix (`android.intent.category.`) der abschließende Teil eines Konstantennamens (in diesem Fall ohne `CATEGORY_`) hinzugefügt.

Weiterführende Informationen zu Intents und Intent-Filtern finden Sie im Dokument *Intents and Intent Filters*.[4]

4 *http://developer.android.com/guide/topics/intents/intents-filters.html*

4.2.2 Kommunikation zwischen Activities

Activities repräsentieren die für den Anwender sichtbaren Teile einer App. Indem er durch die verschiedenen Bereiche der Anwendung navigiert, startet und beendet er unbewusst (je nach Komplexität des Programms) eine ganze Reihe von Activities. Diese rufen ihren jeweiligen Nachfolger auf, indem sie der Methode `startActivity()` ein Intent übergeben, das diesen näher beschreibt. Dies kann eine namentlich bekannte Activity sein. Solche expliziten Intents enthalten den Ihnen bereits bekannten Komponentennamen. Um beispielsweise die Activity `SimpleActivity` zu starten, fügen Sie der aufrufenden Aktivität die folgenden Zeilen hinzu:

```
Intent intent = new Intent(this, SimpleActivity.class);
startActivity(intent);
```

Tipp

Denken Sie daran, dass Sie jede Activity, die Sie mit `startActivity()` aufrufen möchten, in die Manifestdatei eintragen müssen.

Bei impliziten Intents ist der konkrete Klassenname nicht bekannt. Das System sucht stattdessen anhand der *Action* die am besten geeignete Activity aus. Diese kann, wie Sie bereits wissen, auch zu einer fremden App gehören.

Parameter übergeben

Mit dem folgenden Quelltextfragment können Sie im Android-Adressbuch einen neuen Kontakt anlegen. Um es auszuprobieren, erzeugen Sie einfach ein beliebiges neues Projekt und fügen es der automatisch erzeugten Activity-Klasse am Ende der Methode `onCreate()` hinzu.

```
Intent in = new Intent(Intent.ACTION_INSERT,
      ContactsContract.Contacts.CONTENT_URI);
in.putExtra(ContactsContract.Intents.Insert.NAME,
   "Max Mustermann");
in.putExtra(ContactsContract.Intents.Insert.PHONE,
   "+49 (123) 45 67 89");
in.putExtra(ContactsContract.Intents.Insert.PHONE_TYPE,
   ContactsContract.CommonDataKinds.Phone.TYPE_WORK);
startActivity(in);
```

Listing 4.9 Einen neuen Kontakt anlegen

Dem Intent wird die Action `ACTION_INSERT` zugewiesen. Der zweite Parameter des Konstruktoraufrufs legt fest, auf welchem Datenbestand die Aktion ausgeführt wer-

den soll. Die im Beispiel übergebene Konstante verweist auf die zentrale Kontaktdatenbank. Alternativ ist übrigens auch die folgende Schreibweise möglich:

```
Intent in = new Intent(Intent.ACTION_INSERT);
in.setData(ContactsContract.Contacts.CONTENT_URI);
```


Mit der Methode `putExtra()` können Sie einem Intent Nutzdaten übergeben. Was diese bewirken und wie sie ausgewertet werden, ist letztlich von der Activity abhängig, die das Intent empfängt. In der Android-Dokumentation finden Sie entsprechende Beschreibungen bei den `ACTION_`-Konstanten der Klasse `Intent`.

Wenn Sie Ihrer Activity unmittelbar nach dem Aufruf der Methode `startActivity()` eine Ausgabe mittels `Log.d()` hinzufügen, stellen Sie fest, dass Android nicht auf die Beendigung der nachgestarteten Activity wartet. Dies ist aber unter Umständen gewünscht, um entsprechend darauf reagieren zu können. Sicherlich fragen Sie sich, wie Sie ermitteln können, wann dies passiert.

Rückgabewerte

Hierfür ist die Methode `onActivityResult()` zuständig. Sie erhält als zusätzlichen Parameter einen sogenannten *Request Code*. Hierbei handelt es sich um einen Wert, den Sie beliebig vergeben können. Sie kennzeichnen mit ihm, *wer* eine Activity gestartet hat. Es bietet sich an, in der aufrufenden Klasse eine entsprechende Konstante zu definieren. Außerdem überschreiben Sie die Methode `onActivityResult()`.

Android liefert beim Aufruf der Methode `onActivityResult()` den Request Code zurück, der an `startActivityForResult()` übergeben wurde. Auf diese Weise können von einer Activity aus beliebig viele »Nachfolger« gestartet werden.

```
@Override
protected void onActivityResult(int requestCode,
      int resultCode, Intent data) {
   if (requestCode == RQ_INSERT_CONTACT) {
      Log.d(TAG, resultCode + ", " + data);
      if (resultCode == RESULT_OK) {
         if (data != null) {
            Intent i =
               new Intent(Intent.ACTION_VIEW, data.getData());
            startActivity(i);
         }
      }
   }
}
```

Listing 4.10 Einen Datensatz anzeigen

Wichtig ist nur, ihnen eindeutige Request Codes zuzuordnen. Diese werten Sie einfach in `onActivityResult()` aus. Auch für Rückantworten von nachgestarteten Activities verwendet Android Intents. Beispielsweise liefert die Activity zum Anlegen von Kontakten einen URI, der den neuen Eintrag kennzeichnet. Mein Beispiel verwendet diese Referenz, um den Datensatz anzuzeigen. Denken Sie daran, vor Zugriffen das Intent auf `null` zu prüfen. Activities müssen nämlich nicht zwingend eine Nachricht zurückliefern.

4.2.3 Broadcast Receiver

In diesem Abschnitt stelle ich Ihnen einen weiteren wichtigen Anwendungsbaustein vor. *Broadcast Receiver* sind Komponenten, die auf systemweit versandte Nachrichten reagieren. Android verschickt solche Mitteilungen zum Beispiel, wenn der Batteriestand niedrig ist oder der Bildschirm ausgeschaltet wurde. Auch für normale Apps kann es interessant sein, solche *Broadcasts* zu initiieren. Denken Sie an ein Programm für Dateitransfers. Dieses möchte sicherlich darüber informieren, dass eine Übertragung (erfolgreich oder mit Fehler) abgeschlossen wurde.

Broadcast Receiver haben keine eigene Bedienoberfläche, können aber Benachrichtigungen in der Statuszeile hinterlassen. Generell sind sie jedoch eher Schnittstellen zu anderen Komponenten. Insofern sollten Ihre Broadcast Receiver so wenig Logik wie möglich enthalten und diese an Activities oder Services delegieren. Übrigens kann Android diese Bausteine unabhängig von anderen Komponenten einer App aktivieren.

Broadcast Receiver implementieren

Das folgende Beispiel zeigt einen einfachen Broadcast Receiver, der nach jedem Systemstart eine Meldung in der Statuszeile hinterlässt. Sie finden das Projekt *BroadcastReceiverDemo* auf der Begleit-DVD des Buches im gleichnamigen Unterverzeichnis von *Quelltexte*.

```
package com.thomaskuenneth.broadcastreceiverdemo;

import java.text.DateFormat;
import java.util.Date;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
public class BootCompletedReceiver extends BroadcastReceiver {

   private static final int ID = 42;

   @Override
   public void onReceive(Context context, Intent intent) {
      if (Intent.ACTION_BOOT_COMPLETED.equals(
            intent.getAction())) {
         // Notification
         String msg =
            DateFormat.getDateTimeInstance().format(
            new Date());
         Notification n =
            new Notification(R.drawable.icon, msg,
            System.currentTimeMillis());
         // Intent und PendingIntent bauen
         Intent contentIntent = new Intent(context,
               BroadcastReceiverDemo.class);
         n.setLatestEventInfo(context,
               context.getString(R.string.text),
               msg,
               PendingIntent.getActivity(
               context, 0, contentIntent,
               Intent.FLAG_ACTIVITY_NEW_TASK));
         // Mitteilung anzeigen
         NotificationManager m =
               (NotificationManager) context
         getSystemService(Context.NOTIFICATION_SERVICE);
         m.notify(ID, n);
      }
   }
}
```

Listing 4.11 BootCompletedReceiver.java

Broadcast Receiver leiten von der Klasse `android.content.BroadcastReceiver` ab und müssen die Methode `onReceive()` überschreiben. Diese wird aufgerufen, sobald der Receiver eine Nachricht in Form eines Intents empfängt. Dessen Verarbeitung darf allerdings nicht sehr viel Zeit in Anspruch nehmen. Nach zehn Sekunden kann Android den zugehörigen Prozess blockieren und beenden. Beachten Sie auch, dass das Receiver-Objekt nach dem Verlassen von `onReceive()` aus dem Speicher entfernt werden kann. Asynchron gestartete Operationen sollten ihm deshalb keine Ergebnisse übermitteln.

Ausführliche Hinweise zur Kommunikation mit Services finden Sie in Kapitel 6, »Multitasking«. Übrigens weist die Entwicklerdokumentation darauf hin, dass nicht nur Intent-Filter, die in der Manifestdatei eingetragen oder programmatisch mittels `registerReceiver()` gesetzt wurden, an einen Receiver übermittelt werden können. Aus diesem Grund sollten Sie, wie in `BootCompletedReceiver` zu sehen ist, ausdrücklich auf alle erwarteten Aktionen prüfen.

Broadcast Receiver in der Manifestdatei definieren

Damit Android den Broadcast Receiver über einen abgeschlossenen Systemstart informiert, muss die zugehörige App die Berechtigung `android.permission.RECEIVE_BOOT_COMPLETED` anfordern. Dies geschieht mit einem entsprechenden `<uses-permission />`-Element in der Manifestdatei. Der Eintrag für den Receiver selbst sieht folgendermaßen aus:

```
<receiver android:name=".BootCompletedReceiver">
   <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED" />
   </intent-filter>
</receiver>
```

Listing 4.12 Einen Broadcast Receiver definieren

Das Element `<receiver />` kann zahlreiche Attribute erhalten. Ein paar von ihnen werden Sie praktisch immer verwenden, andere hingegen eher selten. `android:name` verweist auf die voll qualifizierte Klasse, die den Receiver implementiert. Ist der Paketname leer (beginnt der Eintrag also mit einem Punkt, auf den der Klassenname folgt), wird als Paket das im `<manifest />`-Element spezifizierte verwendet. Dieses Attribut müssen Sie setzen.

`android:enabled` legt fest, ob ein Receiver vom System instanziiert werden kann. Standardmäßig ist das der Fall. `android:exported` regelt, ob ein Receiver Nachrichten von Quellen erhalten kann, die außerhalb der eigenen App liegen. `false` legt fest, dass nur App-interne Komponenten Mitteilungen an ihn senden können. Der Standardwert hängt davon ab, ob ein Receiver Intent-Filter enthält.

Ist dies nicht der Fall, kann er ausschließlich von Intents aufgerufen werden, die den exakten Klassennamen angeben. Dies impliziert, dass der Receiver nur für die anwendungsinterne Kommunikation vorgesehen ist. Deshalb ist in diesem Fall der Standardwert `false`.

Das Vorhandensein von Intent-Filtern hingegen deutet darauf hin, dass ein Receiver Nachrichten des Systems oder von anderen Apps verarbeiten soll. In diesem Fall ist der Standardwert `true`. Mit `android:icon` und `android:label` schließlich können Sie einem Broadcast Receiver individuelle Symbole und Bezeichnungen zuweisen. Feh-

len die Attribute, werden stattdessen die korrespondierenden Einträge der App ausgewertet.

Sie haben bereits das Attribut `android:exported` kennengelernt, mit dem Sie die Nutzbarkeit von Receivern durch App-fremde Komponenten steuern können. Berechtigungen sind eine weitere Möglichkeit, den Zugriff auf sie zu regeln. `android:permission` erhält den Namen einer Berechtigung, die Broadcaster haben müssen, um einem Receiver eine Nachricht senden zu können. Wird dieses Attribut nicht gesetzt, gelten die Berechtigungen, die innerhalb des `<application />`-Elements gesetzt wurden. Ist keines der beiden Attribute gesetzt, ist ein Broadcast Receiver nicht durch eine Berechtigung geschützt.

Das Attribut `android:process` enthält den Namen eines Prozesses, in dem ein Broadcast Receiver ausgeführt werden sollte. Normalerweise laufen alle Komponenten einer App in ihrem Standardprozess, der den Namen des App-Packages trägt.

Mit dem gleichnamigen Attribut des `<application />`-Elements kann für alle Komponenten ein alternativer Standardprozess gesetzt werden. Da alle Bausteine einer App das Attribut angeben können, lässt sich die Anwendung auf mehrere Prozesse verteilen. Wenn sich Komponenten mehrerer Apps einen Prozess teilen, spart dies Ressourcen. Die Ausführung in einem bestimmten Prozess ist natürlich nur möglich, wenn der Broadcast Receiver eine entsprechende Berechtigung hat.

4.3 Fragmente

In diesem Abschnitt stelle ich Ihnen einen weiteren Grundbaustein für Android-Apps vor. *Fragmente* haben mit Version 3.0 zunächst auf Tablets Einzug in die Plattform gehalten und stehen mit *Ice Cream Sandwich* auch auf Smartphones zur Verfügung. Um sie zu nutzen, tragen Sie in der Manifestdatei Ihrer App für `android:minSdkVersion` oder `android:targetSdkVersion` mindestens 11 (*Honeycomb*) ein. Für Geräte mit früheren Android-Versionen stellt Google eine Kompatibilitätsbibliothek zur Verfügung. Sie können das *Android support package* über den Android SDK Manager herunterladen.

4.3.1 Grundlagen

Wie Sie wissen, strukturieren Activities eine Anwendung. Aufgrund der ihnen zugrunde liegenden Idee, nur genau so viel anzuzeigen oder abzufragen, wie zum Erledigen einer ganz bestimmten Aufgabe nötig ist, lassen sie sich hervorragend wiederverwenden. Ich habe Ihnen dies im vorherigen Abschnitt anhand der Eingabe und späteren Darstellung eines Kontakts bereits demonstriert. Im Gegensatz zu Smartphones bieten Tablet-Computer sehr viel Platz zum Anzeigen von Informationen.

Die auf kleinen Bildschirmen notwendige Trennung zwischen Übersicht und Detaildarstellung ist hier nicht mehr nötig, aus Sicht des Benutzers sogar störend.

Hierzu ein Beispiel: Viele Android-Apps enthalten eine Hauptaktivität, die ihre Funktionen oder Module als Menü (oftmals in Gestalt einer scrollbaren Liste) anbietet. Das Antippen einer Funktion startet eine neue Activity, die die gewünschte Operation ausführt. Tippt der Anwender auf ZURÜCK, erscheint wieder die Übersichtsseite. Auf Geräten mit großem Display ist diese zeitliche Abfolge vollkommen unnötig. Es bietet sich an, stattdessen die Modulauswahl am linken Rand des Bildschirms ständig sichtbar zu lassen. Der Benutzer kann jederzeit eine neue Funktion auswählen, ohne das aktuell ausgeführte Modul explizit »verlassen« zu müssen.

Unterschiede zu Views und ViewGroups

Die Ausgestaltung von Benutzeroberflächen konnte durch *alternative Layouts* schon vor Android 3.0 an unterschiedliche Hardwaregegebenheiten angepasst werden. Ausführliche Informationen hierzu finden Sie im folgenden Kapitel. Insofern fragen Sie sich vielleicht, ob das gerade eben beschriebene Mehrspaltenlayout auf Tablets unbedingt neuer Grundbausteine bedarf.

Wie Sie wissen, wird die eigentliche Fachlogik in den Activity-Klassen implementiert. Die Benutzeroberflächenbeschreibungen geben nur darüber Auskunft, welche Elemente an einer bestimmten Position der Anzeige erscheinen sollen. Wenn also eine Layoutdatei sowohl die Menüauswahl als auch modulbezogene Bedienelemente enthält, muss die Activity beide Bereiche mit Inhalt füllen können.

Fragmente sind Komponenten mit eigener Benutzeroberfläche und eigenem Lebenszyklus. Sie sind Bausteine innerhalb von Activities. Sie werden wie Views und ViewGroups entweder in Layoutdateien definiert oder programmatisch erzeugt. Fragmente werden stets im Kontext einer Activity ausgeführt. Sie können nicht losgelöst von ihr verwendet werden. Das bedeutet: Wird eine Activity gestoppt, halten auch ihre Fragmente an. Und das Zerstören einer Activity führt auch zur Zerstörung ihrer Fragmente.

Ein Beispielfragment

Die Klasse `TestFragment1` stellt Ihnen den grundsätzlichen Aufbau von Fragmenten vor. Um sie auszuprobieren, kopieren Sie das Projekt *FragmentDemo1* aus dem Verzeichnis *Quelltexte* der Begleit-DVD in den Eclipse-Arbeitsbereich. Die App *FragmentDemo1* sehen Sie in Abbildung 4.2.

```
package com.thomaskuenneth.fragmentdemo1;

import android.app.Fragment;
import android.os.Bundle;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class TestFragment1 extends Fragment {

   @Override
   public View onCreateView(LayoutInflater inflater,
         ViewGroup container,
         Bundle savedInstanceState) {
      return inflater.inflate(R.layout.fragment_layout,
         container,
         false);
   }

   @Override
   public void onStart() {
      super.onStart();
      TextView tv = (TextView) getView();
      tv.setText(getString(R.string.text1));
   }
}
```

Listing 4.13 TestFragment1.java

Fragmente leiten entweder direkt von der Basisklasse `android.app.Fragment` ab oder von einem ihrer spezialisierten Kinder, beispielsweise `ListFragment` oder `DialogFragment`. `TestFragment` überschreibt zwei Methoden. `onCreateView()` wird aufgerufen, wenn ein Fragment den Komponentenbaum seiner Benutzeroberfläche instanziieren soll.

Die Beispielimplementierung entfaltet hierzu die sehr einfach gehaltene Layoutdatei *fragment_layout.xml*. Sie definiert als einziges Bedienelement eine `TextView`:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

Listing 4.14 fragment_layout.xml

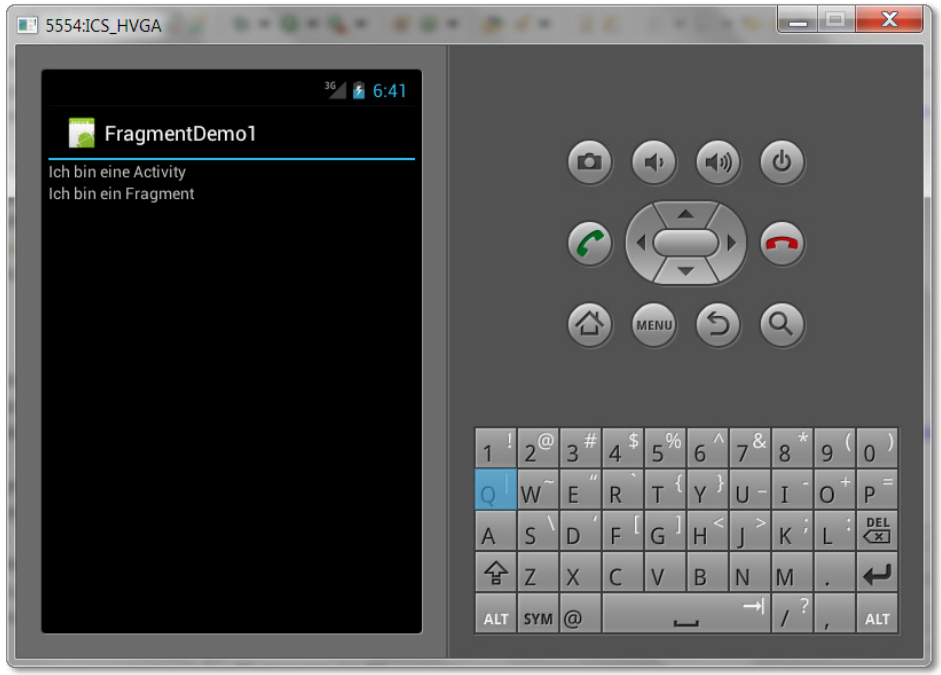


Abbildung 4.2 Die App »FragmentDemo1«

Um etwaige zusätzliche Ressourcen, die beim Instanziieren des Komponentenbaums reserviert wurden, wieder freizugeben, überschreiben Sie gegebenenfalls die Methode `onDestroyView()`. Wenn Sie nur eine XML-Datei entfaltet haben, ist dies aber nicht nötig. Übrigens müssen Fragmente nicht unbedingt eine Benutzeroberfläche haben. Sie können sich das beispielsweise zunutze machen, um auf diese Weise Daten nachzuladen oder um Berechnungen auszuführen. In so einem Fall liefert `onCreateView()` einfach den Wert `null`. Die Basisklasse `Fragment` ist so implementiert (und muss deshalb nicht überschrieben werden).

Die zweite in der Klasse `TestFragment` implementierte Methode `onStart()` wird aufgerufen, wenn ein Fragment für den Benutzer sichtbar wird. Im Beispiel setze ich an `setText()` eine Nachricht, die durch die `TextView` des Fragments angezeigt wird.

4.3.2 Ein Fragment in eine Activity einbetten

Sicherlich fragen Sie sich, wie ein Fragment einer Activity zugeordnet wird. Sehen Sie sich zunächst die Klasse `FragmentDemo1` an. Die Benutzeroberfläche der Activity wird aus der Layoutdatei *main.xml* entfaltet:

```
package com.thomaskuenneth.fragmentdemo1;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class FragmentDemo1 extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = (TextView) findViewById(R.id.textview);
        tv.setText(getString(R.string.text2));
    }
}
```

Listing 4.15 FragmentDemo1.java

main.xml ist folgendermaßen aufgebaut:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
>

   <TextView
      android:id="@+id/textview"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
   />

   <fragment
      android:id="@+id/fragment"
      android:name=
         "com.thomaskuenneth.fragmentdemo.TestFragment1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
   />

</LinearLayout>
```

Listing 4.16 main.xml

Die Datei *main.xml* ordnet eine `TextView` oberhalb eines Fragments an. Die beiden Elemente erhalten die Ids `fragment` und `textview`. Das Attribut `android:name` legt fest, welche Klasse das hier definierte Fragment implementiert.

Lebenszyklus von Fragmenten

Im Gegensatz zu Activities, Services oder Broadcast Receivern tragen Sie Fragmente nicht in die Manifestdatei ein. Sie verbinden sie mit einer Activity, indem Sie Fragmente in deren Benutzeroberfläche integrieren. Dies geschieht, wie Sie gesehen haben, normalerweise deklarativ in Layoutdateien. Insofern verhalten sich Fragmente in diesem Punkt analog zu »normalen« Bedienelementen, die ich Ihnen im folgenden Kapitel ausführlich vorstellen werde. Im Gegensatz zu ihnen haben Fragmente allerdings ein Verhalten – sie implementieren »Fachlogik«. Dies macht sie zu Grundbausteinen von Apps.

Zahlreiche Methoden bestimmen den Lebenszyklus eines Fragments. Beispielsweise wird `onAttach()` aufgerufen, wenn ein Fragment an eine Activity angehängt wurde. Ihr folgt `onCreate()`. Prinzipiell können Sie diese Methode überschreiben, um erste Initialisierungen vorzunehmen. Allerdings muss die zugeordnete Activity zu diesem Zeitpunkt nicht vollständig initialisiert sein. Aus diesem Grund steht zusätzlich `onActivityCreated()` zur Verfügung. Diese Methode wird erst nach der vollständigen Abarbeitung der Activity-Methode `onCreate()` aufgerufen.

Auch am Ende des Lebenszyklus eines Fragments werden zahlreiche Callbacks durchlaufen. `onPause()` signalisiert, dass ein Fragment nicht mehr mit dem Benutzer interagiert (zum Beispiel, weil die gleichnamige Activity-Methode aufgerufen wurde). Nach `onStop()` ist ein Fragment nicht mehr für den Anwender sichtbar. Abschließende Aufräumarbeiten sollten Sie in `onDestroy()` durchführen. `onDetach()` kündigt die Abkopplung eines Fragments von der ihm zugeordneten Activity an.

Komplexe Interaktion mit Activities

Wie Sie gesehen haben, ist der Lebenszyklus von Fragmenten eng mit dem von Activities verknüpft. Und es gibt noch eine weitere Verzahnung zwischen den beiden Grundbausteinen. Fragmente können nämlich in den Zurück-Stapel von Activities integriert werden.

Wenn ein Fragment nicht immer sichtbar ist, sondern nur unter bestimmten Umständen sichtbar wird, ist es für eine durchgängige Bedienung wichtig, dass der Anwender das Fragment auf eine ihm vertraue Weise wieder ausblenden kann. Hierzu hat Google das Konzept der *Fragment-Transaktionen* eingeführt. Sie können zur Laufzeit Fragmente zu Activities hinzufügen oder von ihnen entfernen. Diese Operationen werden zu logischen Schritten zusammengefasst. Jeder dieser Schritte wiederum kann auf den Zurück-Stapel gepackt werden.

Vielleicht fragen Sie sich, wozu das nötig sein soll. Nehmen Sie an, das Auswählen eines Menübefehls führt dazu, dass eine Activity drei zusätzliche Fragmente anzeigt. Drückt der Anwender die ZURÜCK-Taste, möchte er sehr wahrscheinlich nicht jedes Fragment einzeln schließen, sondern alle drei auf einmal. Ausführliche Informationen hierzu finden Sie im Dokument *Fragments*.[5]

5 *http://developer.android.com/guide/topics/fundamentals/fragments.html*

Kapitel 5
Benutzeroberflächen

Die Bedienoberfläche ist das Aushängeschild einer App. Gerade auf mobilen Geräten ist es wichtig, dem Anwender die Nutzung eines Programms so einfach wie möglich zu machen. Dass dabei der Spaß nicht auf der Strecke bleiben muss, zeige ich Ihnen in diesem Kapitel.

Android stellt Ihnen eine ganze Reihe von Bedienelementen zur Verfügung, mit denen Sie die Benutzeroberfläche Ihrer App gestalten können. Diese Komponenten werden zur Laufzeit zu einem Objektbaum entfaltet. Anders als beispielsweise in Java Swing, wo Hierarchiebeziehungen zwischen Komponenten und Objekteigenschaften auf Quelltextebene modelliert werden müssen, nutzt Android hierfür XML.

5.1 Views und ViewGroups

Sie haben Layoutbeschreibungen schon in den vorangegangenen Kapiteln verwendet. Entsprechende Dateien werden in Unterverzeichnissen von *res* abgelegt, die mit *layout* beginnen. Das folgende Beispiel *main.xml* ist Bestandteil des Projekts *Widget-Demo*, das Sie im Ordner *Quelltexte* der Begleit-DVD des Buches finden. Die vollständige App ist in Abbildung 5.1 zu sehen. *main.xml* ist recht einfach gehalten:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="6dp"
>

  <EditText
    android:id="@+id/textfield"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/hint"
```

```
  />

  <Button
    android:id="@+id/apply"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/apply"
  />

  <FrameLayout
    android:id="@+id/frame"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
  />

</LinearLayout>
```

Listing 5.1 main.xml

Schon auf den ersten Blick fällt auf, dass die vier Elemente `LinearLayout`, `FrameLayout`, `Button` und `EditText` einige gemeinsame Attribute haben. `android:layout_width` und `android:layout_height` sind überall vorhanden. `android:id` fehlt nur in `LinearLayout`.

IDs werden benötigt, um sich innerhalb einer Klasse, zum Beispiel einer Activity, auf eine View oder ViewGroup beziehen zu können – vielleicht weil Sie die Farbe setzen oder einen Text ändern wollen. In meinem Beispiel ist dies für `LinearLayout` aber nicht nötig.

Zu jedem XML-Element gibt es ein Pendant in der Android-Klassenbibliothek. Instanzen dieser Klassen bilden zur Laufzeit einer App einen Objektbaum, der die Benutzeroberfläche repräsentiert. Die XML-Attribute werden hierbei auf Instanzvariablen abgebildet. Das System nimmt Ihnen die Arbeit des Entfaltens praktisch vollständig ab. Häufig ist nur die Ihnen bereits bekannte Anweisung `setContentView(...);` notwendig.

5.1.1 Views

Die Basisklasse aller Bedienelemente ist `android.view.View`. Die Benutzeroberfläche einer App besteht also aus einer oder mehreren Views oder von ihr abgeleiteten Klassen. Sie fasst die Eigenschaften und Methoden zusammen, die mindestens nötig sind, um an einer bestimmten Position ein rechteckiges Element mit vorgegebener Größe darzustellen. Beispielsweise wird `onDraw()` aufgerufen, wenn sich die Komponente zeichnen soll. Views sind also für ihr Rendering selbst verantwortlich. Ebenso kümmern sie sich um die Bearbeitung von Tastatur-, Touch- und Trackball-Ereignissen.

Eigenschaften von Views

Wenn die Benutzeroberfläche in einer XML-Datei deklariert und erst zur Laufzeit zu einem Objektbaum entfaltet wird, muss es einer App möglich sein, Referenzen auf spezifische Komponenten zu ermitteln. Dies ist beispielsweise nötig, um auf Benutzeraktionen zu reagieren. Denken Sie an das Anklicken einer Schaltfläche oder unmittelbare Reaktionen auf Eingaben in ein Textfeld.

Activities enthalten hierfür die Methode `findViewById()`. Der ihr übergebene Wert entspricht üblicherweise einer Konstante aus `R.id`. Diese wiederum bezieht ihre Informationen aus den Attributen `android:id` der XML-Dateien. Das Eingabefeld meines Beispiels ist über die ID `textfield` erreichbar. Die Klasse `WidgetDemo`, die Sie gleich kennenlernen werden, greift mit der Anweisung

```
final EditText e = (EditText) findViewById(R.id.textfield);
```

auf sie zu. Zusätzlich zu diesen IDs können Sie Views ein *Tag* zuweisen. Tags werden nicht für die Identifizierung von Views verwendet, sondern dienen als eine Art Speicher für Zusatzinformationen. Anstatt solche Daten in einer gesonderten Struktur abzulegen, können Sie diese (oder gegebenenfalls eine Referenz auf sie) direkt in der View ablegen. In Kapitel 3, »Von der Idee zur Veröffentlichung«, nutzt die Methode `getView()` der Klasse `TierkreiszeichenAdapter` Tags, um Referenzen auf `convertView`s zu speichern.

Views sind Rechtecke. Ihre Positionen werden durch ihre linken oberen Ecken bestimmt. Die Werte entsprechen Pixeln. Sie können sie mit `getLeft()` und `getTop()` abfragen.

Hinweis

Views bilden Hierarchien ab. Koordinaten werden deshalb stets relativ zur Elternkomponente angegeben, nicht als absolute Werte.

Die Größe einer View ergibt sich aus den Dimensionen Breite und Höhe. Tatsächlich gehören sogar zwei solcher Paare zu einer View. Das erste gibt an, wie groß innerhalb seines Elternobjekts es sein möchte. Die beiden Dimensionen lassen sich mit `getMeasuredWidth()` und `getMeasuredHeight()` abfragen.

Das zweite Paar gibt die tatsächliche Größe einer View auf dem Bildschirm an, und zwar nach dem Layoutvorgang, aber vor dem Zeichnen. Diese Werte werden von den beiden Methoden `getWidth()` und `getHeight()` geliefert. Die beiden Paare können, müssen aber nicht unterschiedlich sein.

Die Größe einer View wird auch durch das sogenannte *Padding* beeinflusst. Diese Pixelwerte geben an, wie weit der Inhalt einer View von ihrem oberen, unteren, linken und rechten Ende entfernt sein soll. Das Padding können Sie mittels `setPad-`

ding() setzen und mit getPaddingLeft(), getPaddingTop(), getPaddingRight() und getPaddingBottom() abfragen.

Das Konzept von Rändern wird übrigens in den sogenannten *ViewGroups* umgesetzt. Diese stelle ich Ihnen im folgenden Abschnitt ausführlich vor.

Komponenten programmatisch erzeugen

Die App *WidgetDemo*, die Sie in Abbildung 5.1 sehen, besteht im Wesentlichen aus einem Eingabefeld und einer Schaltfläche.

Abbildung 5.1 Die App »WidgetDemo«

Geben Sie den voll qualifizierten Klassennamen eines Bedienelements (zum Beispiel android.widget.AnalogClock, android.widget.RatingBar oder android.widget.DatePicker) ein, und klicken Sie anschließend auf Übernehmen. Das Programm instanziiert das entsprechende Objekt und fügt es in den Komponentenbaum ein, der aus der XML-Datei *main.xml* entfaltet wurde.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    final LayoutParams params = new
            LayoutParams(LayoutParams.FILL_PARENT,
                LayoutParams.WRAP_CONTENT);
```

```
    // Referenzen auf Views ermitteln
    final FrameLayout f = (FrameLayout) findViewById(R.id.frame);
    final EditText e = (EditText) findViewById(R.id.textfield);
    final Button b = (Button) findViewById(R.id.apply);
    b.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            String name = e.getText().toString();
            try {
                // ein Objekt instanziieren
                Class c = Class.forName(name);
                Object o =
                   c.getDeclaredConstructor(Context.class)
                   .newInstance(WidgetDemo.this);
                if (o instanceof View) {
                    // alte Views löschen
                    // und neue hinzufügen
                    f.removeAllViews();
                    f.addView((View) o, params);
                    f.forceLayout();
                }
            } catch (Throwable tr) {
                Log.e(TAG,
                   "Fehler beim Instanziieren von " +
                   name, tr);
            }
        }
    });
}
```

Listing 5.2 Auszug aus WidgetDemo.java

`FrameLayout` ist ein Kind der Klasse `android.view.ViewGroup`. Solche Container nehmen weitere Bedienelemente auf. Zunächst wird mit der Anweisung

```
FrameLayout f = (FrameLayout) findViewById(R.id.frame);
```

eine Referenz auf die Komponente ermittelt. Anschließend können Sie mit `addView()` Views (oder ViewGroups) hinzufügen und mit `removeAllViews()` entfernen. Wie Sie sehen, ist das Mischen von deklarativ und programmatisch erstellten Oberflächen problemlos möglich.

Auf Benutzeraktionen reagieren

Die Idee, mithilfe von sogenannten *Listenern* auf Benutzeraktionen zu reagieren, wird in vielen Klassenbibliotheken umgesetzt. Bedienelemente bieten an, Referenzen von Objekten, die entweder bestimmte Interfaces implementieren oder von bestimmten Basisklassen ableiten, bei sich zu registrieren. Tritt ein festgelegtes Ereignis ein oder löst der Benutzer eine bestimmte Aktion aus, ruft die Komponente eine Methode der ihr übergebenen Objekte auf.

Welche Listener ein Bedienelement anbieten, entnehmen Sie bitte Googles Entwicklerdokumentation. Der hier gezeigte `OnClickListener` steht zum Beispiel auch bei Ankreuzfeldern (`android.widget.CheckBox`) zur Verfügung.

```
Button b = (Button) findViewById(R.id.apply);
b.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        ...
    }
}
```

Listing 5.3 Auf das Anklicken einer Schaltfläche reagieren

Bei der Wahl des Listeners müssen Sie genau überlegen, was Sie in Ihrer App erreichen möchten. Hierzu ein Beispiel. Das in Abbildung 5.2 gezeigte Programm *ListenerDemo* gibt in einem Textfeld den Status einer `CheckBox` aus. Da die Klasse einen *OnClickListener* registriert, wird die Meldung beim Anklicken der Komponente aktualisiert. Sie können dies ausprobieren, indem Sie das Projekt *ListenerDemo* aus dem Verzeichnis *Quelltexte* der Begleit-DVD des Buches in Ihren Eclipse-Arbeitsbereich kopieren.

```
final TextView textview = (TextView) findViewById(R.id.textview);
final CheckBox checkbox = (CheckBox) findViewById(R.id.checkbox);
checkbox.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
    textview.setText(Boolean.toString(checkbox.isChecked()));
    }
});
final Button status = (Button) findViewById(R.id.status);
status.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        checkbox.setChecked(!checkbox.isChecked());
    }
});
```

Listing 5.4 Auszug aus ListenerDemo.java

Abbildung 5.2 Die App »ListenerDemo«

Die Schaltfläche STATUS ÄNDERN kehrt den aktuellen Zustand des Ankreuzfeldes mit der Anweisung

```
checkbox.setChecked(!checkbox.isChecked());
```

um. Die Komponente selbst reagiert auf die Änderung und zeichnet sich neu. Das Textfeld hingegen wird natürlich nicht aktualisiert. Dies geschieht nur, wenn die CheckBox angeklickt wird.

Eine – allerdings unschöne – Lösung dieses Problems wäre, die Anweisung

```
textview.setText(Boolean.toString(checkbox.isChecked()));
```

in den Listener der Schaltfläche zu kopieren. Zielführender ist, anstelle des OnClick-Listener einen *OnCheckedChangeListener* zu registrieren. Um dies auszuprobieren, entfernen Sie die Anweisung `checkbox.setOnClickListener(...)` und fügen stattdessen den folgenden Ausdruck ein:

```
checkbox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
            boolean isChecked) {
                textview.setText(
```

```
                    Boolean.toString(checkbox.isChecked()));
        }
});
```

Listing 5.5 Auf das Setzen bzw. Entfernen von Häkchen reagieren

Nun verhält sich die App, wie erwartet. Das Anklicken der Schaltfläche führt zur Aktualisierung des Textfeldes, weil aufgrund der Zustandsänderung (ein gesetztes Häkchen wird entfernt und umgekehrt) der `OnCheckedChangeListener` aufgerufen wird.

Im folgenden Abschnitt stelle ich Ihnen sogenannte *ViewGroups* vor. Ähnlich wie beispielsweise Java Swing ordnet Android Bedienelemente auf Grundlage von Regeln an, die in *Layouts* implementiert wurden. Diese Klassen leiten von `android.view.ViewGroup` ab.

5.1.2 Positionierung von Bedienelementen mit ViewGroups

ViewGroups sind Container, können also weitere Views und ViewGroups enthalten. In Ihren Apps verwenden Sie normalerweise nicht diese Basisklasse, sondern abgeleitete Implementierungen wie das Ihnen bereits bekannte `LinearLayout` oder das sehr einfache `FrameLayout`.

FrameLayout und LinearLayout

In der App *WidgetDemo* dieses Kapitels habe ich in der zugehörigen Layoutdatei ein `FrameLayout` definiert, um eine zur Laufzeit erzeugte Komponente in den Objektbaum einhängen zu können. Das ist auch der Haupteinsatzbereich dieser Klasse.

FrameLayouts enthalten normalerweise ein Element, wenngleich sowohl programmatisch mittels `addView()` als auch in der Layoutdatei mehrere Views hinzugefügt werden können. Seine Größe entspricht dem größten Kindelement. Dessen Position wird mit `gravity` kontrolliert. Abbildung 5.3 zeigt die entfaltete Layoutdatei *framelayout_demo.xml*.

Sie finden die Datei im Unterverzeichnis *Diverses* des Ordners *Quelltexte* auf der Begleit-DVD. Um sie auszuprobieren, legen Sie ein neues Android-Projekt nebst Activity an und kopieren die Datei nach *res/layout*. Denken Sie daran, in `onCreate()` den Text `R.layout.main` durch `R.layout.framelayout_demo` zu ersetzen.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```

```
    android:background="@android:color/darker_gray"
>

  <View
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:layout_gravity="top|left"
    android:background="#ffff00"
  />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:background="@android:color/white"
    android:text="@string/hello"
    android:layout_gravity="center"
  />

</FrameLayout>
```

Listing 5.6 framelayout_demo.xml

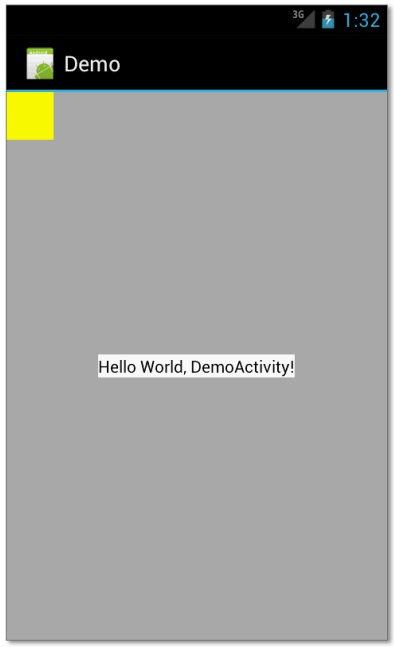


Abbildung 5.3 Die entfaltete Layoutdatei framelayout_demo.xml

Die Kinder werden in der Reihenfolge gezeichnet, in der sie hinzugefügt wurden. Die `TextView` erscheint also am weitesten vorn. Der Wert `center` ihres Attributs `android:layout_gravity` zentriert es. Die mittlere Komponente, eine `View`, erscheint als solides gelbes Rechteck, weil ich mit dem Ausdruck `android:background="#ffff00"` die Hintergrundfarbe auf gelb gesetzt habe. Dem Hashzeichen folgen drei hexadezimale Zahlen, die die Rot-, Grün- und Blauanteile der zu verwendenden Farbe repräsentieren.

`android:layout_gravity="top|left"` sorgt dafür, dass das Element in der linken oberen Ecke des `FrameLayout` gezeichnet wird. Die Größe der View habe ich als sogenannte *Density-independent Pixels* angegeben. Die Einheit `dp` abstrahiert von der Pixeldichte des verwendeten Bildschirms. Auf diese Weise passen sich Ihre Apps besser an unterschiedliche Anzeigen an. Ausführliche Informationen hierzu finden Sie im Abschnitt 5.2.2, »Bitmaps und Pixeldichte«.

`LinearLayout` ist wahrscheinlich das am häufigsten verwendete Layout. Seine Funktionsweise ist leicht nachvollziehbar, und es lässt sich flexibel einsetzen. Es ordnet seine Kinder neben- oder untereinander an. Sie können deren Ausrichtung mit der Ihnen bereits bekannten `gravity` kontrollieren.

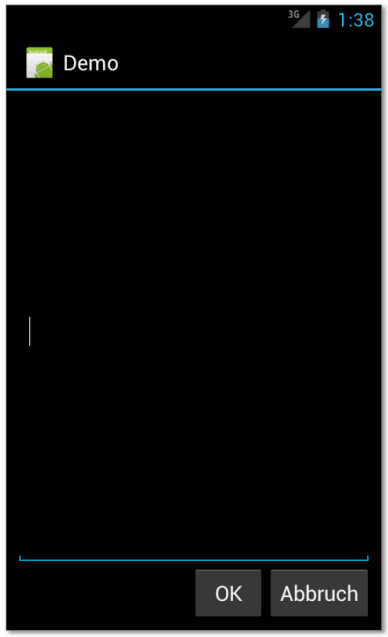


Abbildung 5.4 Die entfaltete Layoutdatei linearlayout_demo.xml

Die in Abbildung 5.4 dargestellte Datei *linearlayout_demo.xml* zeigt im entfalteten Zustand zwei Schaltflächen und ein Eingabefeld. Sie finden die Datei ebenfalls im Unterverzeichnis *Diverses* des Ordners *Quelltexte* auf der Begleit-DVD.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="8dp"
>

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1.0"
    />

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="right"
    >

        <Button
            android:text="OK"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        />

        <Button
            android:text="Abbruch"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        />
    </LinearLayout>
</LinearLayout>
```

Listing 5.7 linearlayout_demo.xml

Mit dem Attribut `android:orientation` steuern Sie, ob Kindelemente spalten- (`horizontal`) oder zeilenweise (`vertical`) angeordnet werden. Der Ausdruck `android:gravity="right"` führt dazu, dass die beiden Schaltflächen des Beispiels gegen den rechten Rand hin ausgerichtet werden.

Freien Bildschirmplatz können Sie mit dem Attribut `android:layout_weight` verteilen. Das funktioniert folgendermaßen: Zunächst ermittelt Android, wie hoch oder breit die Kinder sein möchten. Anschließend wird noch vorhandener Raum entsprechend der im Attribut `layout_weight` angegebenen Gewichtung verteilt. Soll beispielsweise das Eingabefeld zwei Drittel der Bildschirmhöhe einnehmen, setzen Sie dessen weight auf `0.7` und das Attribut `android:layout_weight` des `LinearLayout` der Schaltflächen auf `0.3`. Die Summe aller Gewichtungen muss standardmäßig `1.0` ergeben. Dieser Wert kann bei Bedarf durch Setzen des Attributs `android:weightSum` geändert werden.

Hierarchy Viewer und RelativeLayout

Wie Sie gesehen haben, können Sie durch das Schachteln von LinearLayouts mit geringem Aufwand die Benutzeroberfläche Ihrer App gestalten. Allerdings belegt jede View und jede ViewGroup Speicher. Zudem ist das Entfalten von komplexen Komponentengeflechten aufwendig, verbraucht also Rechenleistung. Auch wenn moderne Smartphones und Tablets mittlerweile eine beachtliche Leistungsfähigkeit erreicht haben, ist effiziente Programmierung weiterhin sehr wichtig.

Das Android SDK enthält deshalb die Anwendung *hierarchyviewer*. Mit diesem Tool, das Sie in Abbildung 5.5 sehen, können Sie den Komponentenbaum einer App live inspizieren. Es verdeutlicht, aus wie vielen Objekten das doch recht einfach anmutende `LinearLayout`-Beispiel besteht. Wenn Sie das Programm starten, während eine App im Emulator ausgeführt wird, können Sie direkt auf den aktuell angezeigten Komponentenbaum zugreifen.

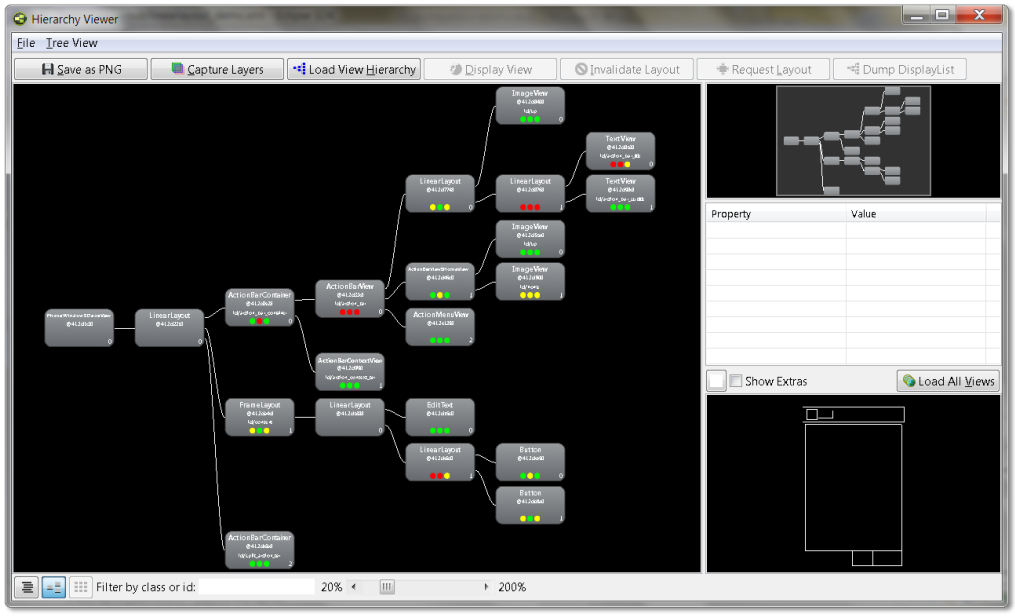


Abbildung 5.5 Der Hierarchy Viewer

Im Folgenden zeige ich Ihnen, wie Sie mit weniger Komponenten dasselbe Ergebnis erzielen. Das `RelativeLayout` kann nicht nur nebeneinanderliegende Schaltflächen ohne zusätzlichen Container anordnen. Die vollständige Benutzeroberfläche des Beispiels aus dem vorherigen Abschnitt ist damit intuitiv und effizient umsetzbar:


```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="8dp"
>

    <Button
        android:id="@+id/button_cancel"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"
        android:text="Abbruch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />

    <Button
        android:id="@+id/button_ok"
        android:layout_toLeftOf="@id/button_cancel"
        android:layout_alignTop="@id/button_cancel"
        android:text="OK"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />

    <EditText
        android:layout_above="@id/button_ok"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />

</RelativeLayout>
```

Listing 5.8 relativelayout_demo.xml

Sie finden die Datei *relativelayout_demo.xml* ebenfalls im Unterverzeichnis *Diverses* des Ordners *Quelltexte* auf der Begleit-DVD. Die Grundidee von `RelativeLayout` ist, die Positionen von Komponenten in Abhängigkeit zu anderen Elementen zu

beschreiben. Beispielsweise sorgt `android:layout_toLeftOf="@id/button_cancel"` dafür, dass die Schaltfläche OK links neben ABBRUCH platziert wird. Ähnliches gilt für `android:layout_above="@id/button_ok"`. Dieser Ausdruck positioniert das Eingabefeld über den beiden Schaltflächen.

Vielleicht fragen Sie sich, warum ich die Benutzeroberfläche sozusagen von unten nach oben beschreibe. Wäre es nicht einfacher, mit dem `EditText` zu beginnen und im Anschluss daran die beiden Buttons zu definieren? Grundsätzlich ist dies möglich, allerdings kann Android dann nicht ohne Weiteres berechnen, wie hoch das Eingabefeld werden darf.

Tipp

Oftmals müssen Sie etwas tüfteln, bis Sie das gewünschte Ergebnis mit `RelativeLayout` erreichen. Die geringere Anzahl von Objekten zur Laufzeit ist diese Mühe meiner Meinung nach in jedem Fall wert.

5.2 Alternative Ressourcen

Wenn man sich die mittlerweile unüberschaubare Vielfalt an Android-Geräten vergegenwärtigt, mag man kaum glauben, dass es im Frühjahr 2009 gerade einmal zwei Modelle gab, die sich zudem (mit Ausnahme einer Tastatur) kaum unterschieden. Für Entwickler war das natürlich eine komfortable Situation. Es gab kaum Unwägbarkeiten, die die Lauffähigkeit der eigenen App gefährdeten. Das Drehen des Displays war beherrschbar.

5.2.1 Automatische Layout-Auswahl

Sowohl das *G1* als auch das *Magic* besaßen ein 3,2 Zoll-Display, das 320 × 480 Pixel auflöste. Drehte der Benutzer das Gerät, oder öffnete er die Hardwaretastatur des G1, wechselte das Seitenverhältnis vom Hochkant- in das Querformat. Diese Funktionalität ist selbstverständlich auch in den aktuellen Android-Versionen enthalten. Auch Tablets profitieren von solchen Orientierungswechseln. Ihre Apps können mit *alternativen Ressourcen* darauf reagieren.

Auf das Drehen des Geräts reagieren

Die Kernidee ist, die Benutzeroberfläche in unterschiedlichen Ausprägungen zur Verfügung zu stellen. Hierzu ein Beispiel. Android beinhaltet das Bedienelement `DatePicker`. Mit ihm lässt sich sehr schnell eine App bauen, die die Anzahl der Tage zwischen zwei Datumsangaben berechnet. Sie ist in Abbildung 5.6 zu sehen. Um das Programm auszuprobieren, kopieren Sie das Projekt *Datumsdifferenz* aus dem Ver-

zeichnis *Quelltexte* der Begleit-DVD in Ihren Eclipse-Arbeitsbereich und importieren es anschließend. Das folgende Listing zeigt dessen Layoutdatei *main.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    >

<DatePicker
    android:id="@+id/date1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false"
    />

<DatePicker
    android:id="@+id/date2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false"
    />

<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >

<Button
    android:id="@+id/button_calc"
    android:text="@string/calc"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />

<TextView
    android:id="@+id/textview_result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/button_calc"
```

```
    android:layout_alignBaseline="@id/button_calc"
    android:layout_marginLeft="16dp"
    />

</RelativeLayout>
</LinearLayout>
```

Listing 5.9 main.xml

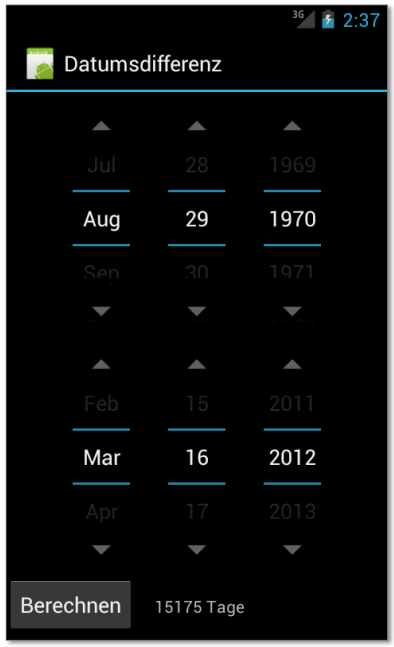


Abbildung 5.6 Die App »Datumsdifferenz«

Nach dem Start bringen Sie den Emulator in den Quermodus, indem Sie auf dem Ziffernblock Ihrer Tastatur [7] (alternativ [Strg] + [F11]) oder [9] ([Strg] + [F12]) drücken. Wie Sie in Abbildung 5.7 sehen, ist das Ergebnis mehr als unbefriedigend, da die Benutzeroberfläche nicht vollständig dargestellt wird.

Um dies zu korrigieren, legen Sie im *Package Explorer* unterhalb von *res* das Verzeichnis *layout-land* an. Es nimmt Layoutdateien auf, die angezeigt werden, wenn das Gerät in den *Landscape*-Modus gebracht wird.

Legen Sie in diesem Ordner eine Datei mit Namen *main.xml* an, und übernehmen Sie die folgenden Zeilen. Um sich Tipparbeit zu ersparen, können Sie stattdessen die Datei *main_land.xml* öffnen, die Sie im Verzeichnis *vorlagen* finden.

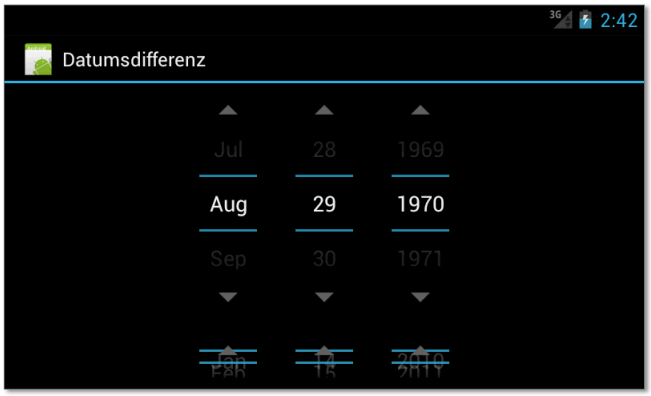


Abbildung 5.7 Die App »Datumsdifferenz« im Quermodus

Fügen Sie ihren Inhalt in *res/layout-land/main.xml* ein.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >

<DatePicker
    android:id="@+id/date1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    />

<DatePicker
    android:id="@+id/date2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@id/date1"
    />

<Button
    android:id="@+id/button_calc"
    android:text="@string/calc"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@id/date2"
    />

<TextView
    android:id="@+id/textview_result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/button_calc"
    android:layout_alignBaseline="@id/button_calc"
    android:layout_marginLeft="16dp"
    />

</RelativeLayout>
```

Listing 5.10 main.xml für den Landscape-Modus

Der Name der Layoutdatei ist für den Portrait- und Landschaftsmodus also gleich, beispielsweise *main.xml*. Dies ist nötig, weil er die Grundlage für eine Konstante in `R.layout` bildet. Diese wiederum verwenden Sie beispielsweise in `setContentView()`. Entscheidend ist, ob Sie die Layoutdatei unter *layout-land* oder *layout* ablegen. Layouts, die ausdrücklich für den Hochkantmodus entworfen wurden, deponieren Sie in *layout-port*.

Android sucht im Ordner *layout*, wenn das angeforderte Layout in keinem der genannten Spezialverzeichnisse gefunden wird. Ausführliche Informationen finden Sie im Entwicklerdokument *Providing Resources*.[1]

Tipp

Im Hinblick auf die Größe der *.apk*-Datei Ihrer App sollten Sie genau prüfen, in welchen Fällen Sie alternative Ressourcen benötigen. Legen Sie Standard-Layouts in *layout* ab, und nutzen Sie Spezialverzeichnisse nur im Bedarfsfall.

Bildschirmgröße

Wie Sie bereits wissen, waren sich die ersten verfügbaren Android-Geräte sehr ähnlich. Das erste Mal mit *Donut*, später mit Einführung von Tablets, hielt die Unterstützung von unterschiedlichen Anzeigegrößen und physikalischen Auflösungen Einzug in die Plattform. Um diese Vielfalt für den Entwickler beherrschbar zu machen, ord-

1 *http://developer.android.com/guide/topics/resources/providing-resources.html*

net Android jedes Gerät in Bezug auf seine Bildschirmdiagonale sowie die Anzahl der darstellbaren Pixel jeweils einer Kategorie zu. Bevor ich sie Ihnen vorstelle, möchte ich noch ein paar Begriffe erklären:

- Die *Bildschirmgröße* wird üblicherweise in Zoll angegeben. Sie beschreibt den Abstand von der linken unteren zur rechten oberen Ecke der Anzeige.
- Das *Seitenverhältnis* (engl. *aspect ratio*) entspricht dem Quotienten aus physikalischer Breite und Höhe. Android kennt in diesem Zusammenhang die beiden Resource-Bezeichner `long` und `notLong`. Ob ein Bildschirm lang oder nicht lang ist, hat übrigens nichts mit dessen Ausrichtung zu tun. WVGA (800 × 480 Pixel) und FWVGA (854 × 480 Pixel) sind lang, VGA (640 × 480 Pixel) hingegen nicht. Deshalb bleibt das Seitenverhältnis zur Laufzeit auch stets gleich.
- Die *Auflösung* gibt die Zahl der horizontal und vertikal ansprechbaren physikalischen Pixel an.
- Die *Pixeldichte* schließlich wird in Punkten pro Zoll angegeben. Dieser Wert ist letztlich ein Maß für die Größe eines Pixels. Er errechnet sich aus der Bildschirmgröße und der physikalischen Auflösung.

Android kennt vier Bildschirmgrößen: klein (ungefähr 2 bis 3,2 Zoll), normal (zwischen 3,2 und ca. 4 Zoll), groß (ungefähr 4 bis 7 Zoll) und sehr groß (derzeit zwischen 7 und 10 Zoll). Entsprechende Layouts werden in den Verzeichnissen *layout-small*, *layout-normal*, *layout-large* und *layout-xlarge* abgelegt. Um ein speziell für den Quermodus entwickeltes Layout für kleine Displays ablegen zu können, erzeugen Sie einfach den Ordner *layout-small-land* und legen es dort ab.

Tipp

Ich rate Ihnen, nur dann spezielle Layouts vorzusehen, wenn es für die Nutzbarkeit Ihrer App erforderlich ist. In vielen Fällen reicht es aus, den Komponentenbaum einer Activity in eine `ScrollView` zu packen. Allerdings funktionieren einige Bedienelemente dann nicht mehr optimal, zum Beispiel die Datumsauswahl `DatePicker`.

Die Klasse `android.widget.ScrollView` leitet von dem Ihnen bereits bekannten `FrameLayout` ab, enthält also normalerweise ein Kind – in diesem Falle Ihr eigentlicher Komponentenbaum. Wenn dieser vollständig angezeigt werden kann, tritt die `ScrollView` nicht weiter in Erscheinung. Auf Geräten mit kleinen Displays hingegen kann der Benutzer bequem zu den normalerweise nicht sichtbaren Elementen navigieren.

Damit Ihre App in Google Play von den passenden Geräten gefunden wird, müssen Sie in der Manifestdatei festlegen, welche Bildschirmkategorien Ihr Programm unterstützt. Je nachdem, welche Werte Sie bei `android:targetSdkVersion` und

`android:minSdkVersion` eingetragen haben, gelten leider unterschiedliche Standardeinstellungen. Deshalb empfehle ich Ihnen, alle passenden Bildschirme explizit aufzuzählen. Hierzu ein Beispiel:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android">
  ...
  <supports-screens
    android:smallScreens="true"
    android:normalScreens="true"
    android:largeScreens="true"
    android:xlargeScreens="false" />
    ...
</manifest>
```

Listing 5.11 Unterstützung für verschiedene Bildschirmgrößen aktivieren

Die Attribute des Elements `<supports-screens />` steuern, für welche Bildschirmgrößen eine App geeignet ist. Im folgenden Abschnitt zeige ich Ihnen, wie Android mit unterschiedlichen Pixeldichten umgeht und welche Auswirkungen dies für die Entwicklung hat.

5.2.2 Bitmaps und Pixeldichte

Beim Erzeugen eines Android-Projekts legt Eclipse (bzw. die Android Development Tools – ADT) unter anderem die Verzeichnisse *drawable-ldpi*. *drawable-mdpi* und *drawable-hdpi* an. Diese enthalten Grafiken (z. B. das Programm-Icon) in unterschiedlichen Pixeldichten. In Abhängigkeit von der Bildschirmkonfiguration des aktuellen Geräts lädt die Plattform zur Laufzeit einer App die gewünschte Datei aus dem Verzeichnis für mittlere (*-mdpi*), niedrige (*-ldpi*), hohe (*-hdpi*) oder sehr hohe (*-xhdpi*) Dichte. Aus diesem Grund sind – wie auch bei Layouts – die eigentlichen Dateinamen stets gleich.

Automatische Skalierung von Bitmaps

Bitmaps, die im Verzeichnis *drawable* (also ohne Postfix) abgelegt werden, skaliert das System zur Laufzeit. Android geht in diesem Fall davon aus, dass solche Grafiken für mittlere Dichte vorgesehen waren. Diese Konvertierung unterbleibt für Dateien im Verzeichnis *drawable-nodpi*.

Bildschirme mit niedriger Dichte stellen etwa 120 Punkte pro Zoll (engl. dots per inch – dpi) dar. Bei mittlerer Dichte sind dies ungefähr 160 dpi und entspricht den beiden ersten Android-Geräten G1 und Magic. Smartphones oder Tablets mit hoher Pixeldichte lösen ca. 240 dpi auf, bei sehr hoher Dichte sind es sogar 320.

Vielleicht fragen Sie sich, warum Android diesen Aufwand treibt. Aus Sicht des Anwenders soll die Benutzeroberfläche gleich groß wirken, unabhängig von technischen Details wie der Pixeldichte. Deshalb werden – wie Sie gleich sehen werden – Größenangaben in Layouts nicht in klassischen Pixeln angegeben und zur Laufzeit entsprechend umgerechnet. Aber gerade Bitmaps verlieren bei jeder Skalierung an Qualität. Aus diesem Grund können Sie Grafiken in für bestimmte Pixeldichten optimierter Form zur Verfügung stellen.

Density-independent Pixels

Physikalische Pixel sind je nach Hardware unterschiedlich groß. Um das Erstellen von Layouts zu vereinfachen, kennt Android deshalb sogenannte *Density-independent Pixels*. Diese abstrakte Einheit basiert auf der Pixeldichte des Bildschirms in Relation zu 160 dpi. `160dp` entsprechen also immer einem Zoll. Die Formel zur Umrechnung ist sehr einfach:

```
pixels = dps * (density / 160);
```

Normalerweise müssen Sie solche Berechnungen in Ihrer App aber gar nicht durchführen. Wichtig ist eigentlich nur, in allen Layouts diese Einheit zu verwenden. Ausführliche Informationen hierzu finden Sie im Abschnitt *Dimension* des Dokuments *More Resource Types*.[2]

5.3 Vorgefertigte Bausteine für Oberflächen

Anwendungsspezifische Teile von Oberflächen setzen Sie aus Views und ViewGroups zusammen. Zu einer »richtigen« App gehören aber auch Dialoge, Einstellungsseiten und Menüs. Diese müssen Sie zum Glück nicht selbst entwickeln, sondern können auf Bausteine der Plattform zurückgreifen.

5.3.1 Nützliche Activities

Sie können die Activities Ihrer App von der Basisklasse `android.app.Activity` ableiten und mit der Anweisung `setContentView(R.layout.main);` eine aus der Datei *main.xml* entfaltete Oberfläche anzeigen. In vielen Fällen ist das aber gar nicht nötig. Android bringt nämlich eine Reihe von bereits abgeleiteten Activities mit.

Listendarstellungen mit der ListActivity

Ein häufig anzutreffendes Gestaltungsmuster in mobilen Apps ist eine listenartige Einstiegsseite, von der aus zu Details verzweigt wird, sobald der Anwender ein Ele-

2 *http://developer.android.com/guide/topics/resources/more-resources.html#Dimension*

ment der Liste antippt. Wie Sie dies mit Android umsetzen können, zeige ich Ihnen anhand der App *MiniContacts* (siehe Abbildung 5.8).

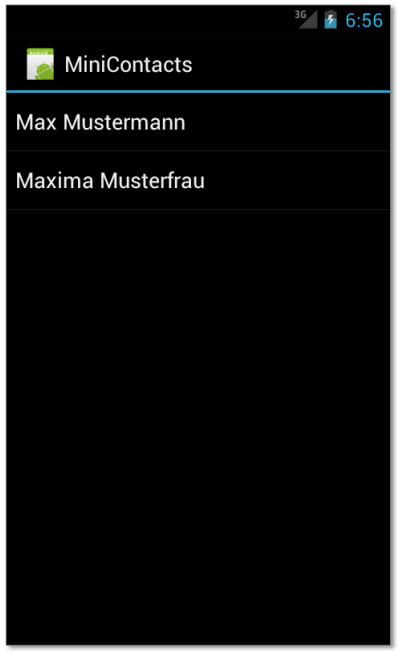


Abbildung 5.8 Die App »MiniContacts«

Das Programm zeigt eine Liste mit den Namen Ihrer Kontakte. Tippen Sie ein Listenelement an, um Details zu einer Person anzuzeigen. Sie finden das Projekt *MiniContacts* im Verzeichnis *Quelltexte* der Begleit-DVD.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Kontaktliste ermitteln
    cursorContacts = getContentResolver().query(
            ContactsContract.Contacts.CONTENT_URI, null,
            null, null, null);
    startManagingCursor(cursorContacts);

    // bildet Datensätze auf Listenelemente ab
    ListAdapter adapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_1,
            cursorContacts,
            new String[] { Contacts.DISPLAY_NAME },
            new int[] { android.R.id.text1 });
```

```
    setListAdapter(adapter);
    getListView().setOnItemClickListener(
        new OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> parent,
                View view,
                int position, long id) {
            Object o = getListAdapter().getItem(position);

            if (o instanceof CursorWrapper) {
                // Welcher Datensatz?
                CursorWrapper w = (CursorWrapper) o;
                int columnIndex =
                    w.getColumnIndex(Contacts._ID);
                long contactId = w.getLong(columnIndex);

                Uri uri = Uri.withAppendedPath(
                    ContactsContract.Contacts.CONTENT_URI,
                    Long.toString(contactId));

                // Kontakt anzeigen
                Intent intent =
                    new Intent(Intent.ACTION_VIEW,
                    uri);
                startActivity(intent);
            }
        }
    });
}
```

Listing 5.12 Auszug aus MiniContacts.java

Die Klasse `android.app.ListActivity` stellt Ihnen eine ListView zur Verfügung. Diese View zeigt ihre Elemente als scrollbare vertikale Liste an. Sie bezieht die darzustellenden Informationen von Adaptern, die das Interface `android.widget.ListAdapter` implementieren. Der im Beispiel verwendete `SimpleCursorAdapter` greift hierzu auf Daten eines Cursors zu. Weiterführende Informationen zu Content Providern finden Sie in Kapitel 11, »Datenbanken«.

Um auf das Antippen eines Listenelements zu reagieren, setzen Sie mit `getListView().setOnItemClickListener()` einen entsprechenden Listener.

Hinweis

Um die Kontaktdatenbank auslesen zu können, müssen Sie in der Manifestdatei mit `<uses-permission />` die Berechtigung `android.permission.READ_CONTACTS` anfordern.

Registerkarten und Reiter

Mit Kartenreitern und Registerkarten lassen sich Informationen elegant strukturieren. Android vereint auf diese Weise Wähldialog, Kontaktliste und Anrufhistorie. Um die hierfür zuständige Klasse `android.app.TabActivity` in Ihrer App zu nutzen, leiten Sie von ihr ab und ermitteln in `onCreate()` zunächst mit

```
TabHost tabHost = getTabHost();
```

die Referenz auf ein Container-Objekt. Dies ist in der Klasse `TabActivityDemo` zu sehen. Diese View `android.widget.TabHost` hat zwei Kinder, die anklickbaren Reiter sowie ein `FrameLayout`, das den eigentlichen Inhalt kapselt.

Registerkarten werden durch Instanzen des Typs `android.widget.TabHost.TabSpec` repräsentiert. Sie bestehen aus einer *Indicator* genannten Anzeige (üblicherweise eine Grafik mit Beschriftung), dem Inhalt sowie einem Tag zur Identifikation. Der Inhalt kann aus einer `Activity` bestehen, die durch ein `Intent` beschrieben wird. Alternativ dürfen Sie auch eine View angeben.

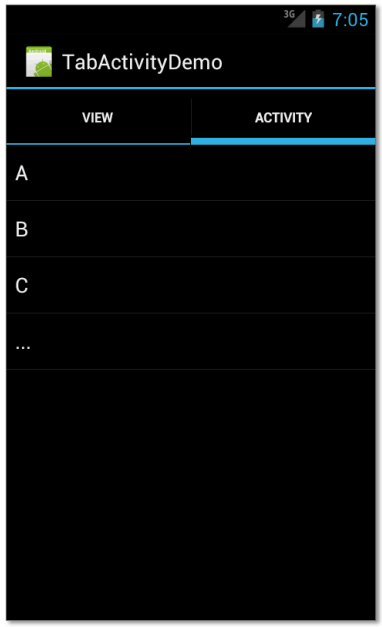


Abbildung 5.9 Die App »TabActivityDemo«

Beide Varianten finden Sie im Projekt *TabActivityDemo*, das ich im Verzeichnis *Quelltexte* der Begleit-DVD abgelegt habe. Die gleichnamige App ist in Abbildung 5.9 zu sehen.

Die dritte Möglichkeit besteht übrigens im Einsatz von `TabHost.TabContentFactory`. Wie dies funktioniert, zeigt die Klasse `com.example.android.apis.view.Tabs2` in Googles *ApiDemos*.


```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    final TabHost tabHost = getTabHost();
    // erstes Tab: eine View
    LayoutInflater.from(this).inflate(R.layout.tab1,
            tabHost.getTabContentView(), true);
    TabSpec ts1 = tabHost.newTabSpec("tab-1");
    ts1.setIndicator(getString(R.string.str_view), getResources()
            .getDrawable(android.R.drawable.ic_secure));
    ts1.setContent(R.id.ll1);
    tabHost.addTab(ts1);
    // zweites Tab: ein Intent
    TabSpec ts2 = tabHost.newTabSpec("tab-2");
    ts2.setIndicator(getString(R.string.str_activity));
    Intent intent = new Intent(this, DemoActivity.class);
    ts2.setContent(intent);
    tabHost.addTab(ts2);
}
```

Listing 5.13 Auszug aus TabActivityDemo.java

TabSpecs können mit der Methode `newTabSpec()` der Klasse `TabHost` instanziiert werden. Den Inhalt einer Registerkarte setzen Sie mit `setContent()`. Die Zeilen nach dem ersten Kommentar zeigen, wie ein Komponentenbaum entfaltet und einem Tab zugewiesen wird. Die zweite Registerkarte zeigt die Activity `DemoActivity` an. Deren minimalen Quelltext finden Sie auf der Begleit-DVD. Durch den Aufruf von `addTab()` werden die fertigen Tabs dem `TabHost` hinzugefügt.

Tipp

Sie können die Action Bar deaktivieren und auf diese Weise die Registerkarten an den oberen Bildschirmrand rücken. Fügen Sie in der Manifestdatei dem Element `<activity />` einfach das Attribut `android:theme="@android:style/Theme.Device-Default.NoActionBar"` hinzu.

Programmeinstellungen mit der PreferencesActivity

Nahezu jede App lässt sich durch den Benutzer anpassen. Wie Sie Android beim Bau solcher Einstellungsseiten unterstützt, zeige ich Ihnen anhand des Programms *PreferencesDemo*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Die App besteht aus den beiden Activities `PreferencesDemo` (die Hauptaktivität) und `MyPreferences`. Abbildung 5.10 zeigt die Einstellungsseite. Sie erscheint, sobald Sie die Schaltfläche EINSTELLUNGEN anklicken.

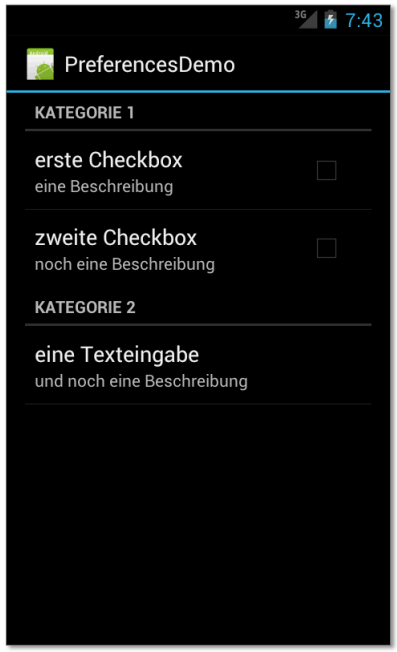


Abbildung 5.10 Die App »PreferencesDemo«

Der Quelltext ist trivial. Die einzige Anweisung, die Sie am besten innerhalb der Methode `onCreate()` ausführen, lautet `addPreferencesFromResource(R.xml.my_preferences);`. Android liest die Datei *my_preferences.xml* aus dem Unterverzeichnis *xml* von *res* ein und baut aus der in ihr enthaltenen Beschreibung einen Komponentenbaum zusammen:

```
<?xml version="1.0" encoding="utf-8"?>

<PreferenceScreen
        xmlns:android="http://schemas.android.com/apk/res/android">

    <PreferenceCategory
            android:title="@string/cat1">
```

```
        <CheckBoxPreference
                android:key="checkbox_1"
                android:title="@string/title_cb1"
                android:summary="@string/summary_cb1" />


        <CheckBoxPreference
                android:key="checkbox_2"
                android:title="@string/title_cb2"
                android:summary="@string/summary_cb2" />

    </PreferenceCategory>

    <PreferenceCategory
            android:title="@string/cat2">

        <EditTextPreference
                android:key="edittext_1"
                android:title="@string/title_et1"
                android:summary="@string/summary_et1"
                android:dialogTitle="@string/dialog_title_et1" />

    </PreferenceCategory>

</PreferenceScreen>
```

Listing 5.14 my_preferences.xml

Mit dem Element `<PreferenceCategory />` können Sie Ihre Einstellungen in Rubriken oder Kategorien unterteilen. Neben den hier gezeigten `<EditTextPreference />` und `<CheckBoxPreference />` kennt Android zahlreiche weitere Elemente, die entweder direkt auf klassische Bedienelemente abgebildet werden oder beim Anklicken Dialoge öffnen – dies ist auch bei `<EditTextPreference />` der Fall.

Sicherlich fragen Sie sich, wie Sie in Ihrem Programm auf gespeicherte Einstellungen zugreifen können. Dies verdeutlicht das folgende Quelltextfragment aus der Klasse `PreferencesDemo`:

```
SharedPreferences prefs = PreferenceManager
        .getDefaultSharedPreferences(this);
boolean cb1 = prefs.getBoolean("checkbox_1", false);
boolean cb2 = prefs.getBoolean("checkbox_2", false);
String et1 = prefs.getString("edittext_1", "");
```

Listing 5.15 Auszug aus PreferencesDemo.java

Das Schlüsselwort `this` bezieht sich auf die Klasse `PreferencesDemo`. Sie leitet von `android.app.Activity` ab und damit indirekt auch von `android.content.Context`. Der erste Parameter der Methoden `getBoolean()` und `getString()` entspricht den Werten, die Sie den Attributen `android:key` in der Datei *my_preferences.xml* zugewiesen haben.

Der zweite wird zurückgeliefert, wenn der angefragte Schlüssel nicht vorhanden ist. In diesem Fall wird er aber nicht angelegt. Um dies auszuprobieren, deinstallieren Sie im Emulator die App *PreferencesDemo* und setzen anschließend die Standardwerte von `cb1` und `cb2` auf `true`. Wenn Sie das Programm erneut starten, werden diese in der erwarteten Weise angezeigt. Ein Klick auf EINSTELLUNGEN zeigt aber, dass die Ankreuzfelder nicht aktualisiert wurden. Um dies zu erreichen, fügen Sie die folgenden Zeilen am Ende des Methodenrumpfes von `updateTextView()` ein:

```
Editor e = prefs.edit();
e.putBoolean("checkbox_1", cb1);
e.putBoolean("checkbox_2", cb2);
e.commit();
```

Listing 5.16 Auszug aus PreferencesDemo.java

Weiterführende Informationen finden Sie im Abschnitt *Using Shared Preferences* des Dokuments *Data Storage*.[3]

5.3.2 Dialoge

Wenn der Benutzer ein `<EditTextPreference />`-Element (um genau zu sein, die daraus erzeugte GUI-Komponente) anklickt, öffnet sich ein kleines Fenster mit einer Überschrift, einem Eingabefeld sowie zwei Schaltflächen. Solche *modalen Dialoge* werden über der aktuellen Activity angezeigt. Sie erhalten den Fokus und nehmen alle Eingaben entgegen. Der Anwender wird also in seiner Tätigkeit unterbrochen. Dialoge sollten deshalb stets in enger Beziehung zu der gegenwärtigen Aktivität stehen. Beispiele sind Aufforderungen zur Eingabe von Benutzername und Passwort, Fortschrittsanzeigen oder Hinweise bzw. Fehlermeldungen.

Lebenszyklus von Dialogen

In der Klasse `DialogDemo` des gleichnamigen Projekts zeige ich Ihnen, wie Sie Dialoge in eigenen Programmen einsetzen. Sie finden die in Abbildung 5.11 dargestellte App im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

3 *http://developer.android.com/guide/topics/data/data-storage.html#pref*

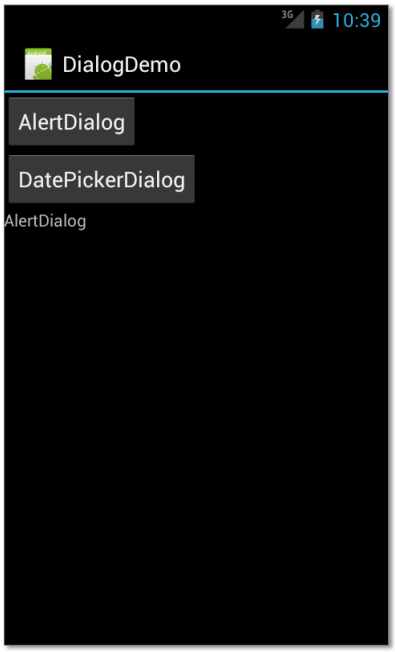


Abbildung 5.11 Die App »DialogDemo«

Nach dem Start sehen Sie die beiden Schaltflächen AlertDialog und DatePickerDialog. Beide öffnen einen Dialog. Die Klasse `android.app.DatePickerDialog` ermöglicht dem Benutzer die Auswahl eines Datums. Android kombiniert hierzu die View `android.widget.DatePicker` mit zwei Schaltflächen zum Übernehmen der Eingabe bzw. dem Abbrechen des Vorgangs.

Damit das System das ausgewählte Datum an Ihre Activity übermitteln kann, registrieren Sie einen `OnDateSetListener`. Ein solches Objekt wird dem Konstruktor von `DatePickerDialog` übergeben. `DatePickerDialog` leitet von `android.app.AlertDialog` ab. Diese Klasse ermöglicht Dialoge mit bis zu drei Schaltflächen, einer Überschrift und einer Nachricht.

`AlertDialog.Builder` macht das Zusammensetzen von solchen Hinweisdialogen sehr einfach. Rufen Sie zunächst die Methoden zum Setzen des Dialogtitels, der Nachricht sowie der Schaltflächen auf, und erzeugen Sie anschließend mit `builder.create()` das eigentliche Dialog-Objekt.

Dialoge werden nach Aufruf der Methode `showDialog()` angezeigt. Der übergebene `int`-Parameter legt fest, welcher gemeint ist. Jeder Dialog, der in einer Activity verwaltet wird, hat seine eigene Nummer. Sie können diese nach Belieben vergeben, müssen aber für ihre Eindeutigkeit sorgen.

In der Methode `onCreateDialog()` bauen Sie Dialoge zusammen. Je nach übergebener ID liefert die Beispielimplementierung entweder eine Instanz der Klasse `android.app.AlertDialog` oder `android.app.DatePickerDialog`.

```
public class DialogDemo extends Activity {
    private static final int DIALOG_ALERT = 1;
    private static final int DIALOG_DATEPICKER = 2;
    private TextView textview;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textview = (TextView) findViewById(R.id.textview);
        Button bAlert = (Button)
            findViewById(R.id.button_alert);
        // auf Anklicken der Schaltfläche reagieren
        bAlert.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                showDialog(DIALOG_ALERT);
            }
        });
        Button bDatePicker = (Button)
            findViewById(R.id.button_datepicker);
        // auf Anklicken der Schaltfläche reagieren
        bDatePicker.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                showDialog(DIALOG_DATEPICKER);
            }
        });
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
        case DIALOG_ALERT:
            // Builder instanziieren
            AlertDialog.Builder builder = new
                AlertDialog.Builder(this);
```

```
            // Builder konfigurieren
            builder.setTitle(R.string.app_name);
            builder.setMessage(R.string.message);
            builder.setCancelable(false);
            builder.setPositiveButton(R.string.close,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog,
                        int id) {
                            textview.setText(
                                getString(R.string.button_alert));
                        }
                    });
            // AlertDialog erzeugen und zurückliefern
            return builder.create();

        case DIALOG_DATEPICKER:
            Calendar cal = Calendar.getInstance();
            OnDateSetListener l = new OnDateSetListener() {
                @Override
                public void onDateSet(DatePicker view,
                    int year,
                    int monthOfYear,
                    int dayOfMonth) {
                        textview.setText(getString(
                            R.string.button_datepicker));
                    }
                };
            DatePickerDialog dp = new DatePickerDialog(this, l,
                    cal.get(Calendar.YEAR), cal.get(Calendar.MONTH),
                    cal.get(Calendar.DAY_OF_MONTH));
            return dp;

        default:
            return super.onCreateDialog(id);
        }
    }
}
```

Listing 5.17 Auszug aus DialogDemo.java

`onCreateDialog()` wird nur einmal aufgerufen. Dies geschieht, bevor ein Dialog zum ersten Mal sichtbar wird. Sie können deshalb zusätzlich die Methode `onPrepareDialog()` überschreiben, um vor jedem Anzeigen aktuelle Werte zu berechnen. Das ist

praktisch, um beispielsweise die gegenwärtige Uhrzeit darzustellen oder aktuelle Messwerte (eines Sensors) zu präsentieren. Weiterführende Informationen zu Dialogen finden Sie im Dokument *Dialogs*.[4]

Dialoge als Activities

Sie haben gesehen, dass Dialoge eng mit Activities verzahnt sind. Dies macht ihren Bau (Überschreiben von maximal zwei Methoden) und ihre Nutzung (Aufruf von `showDialog()`) sehr einfach. Allerdings können Dialoge nicht ohne Weiteres in anderen Activities wiederverwendet werden. Sie müssten hierzu eine Klasse entwickeln, die von `android.app.Dialog` oder `android.app.AlertDialog` ableitet. Jede Activity, die den Dialog nutzen soll, muss `onCreateDialog()` überschreiben und eine Instanz dieser Klasse zurückliefern.

Eleganter ist in diesem Fall, den Dialog als Activity zu implementieren. Auf diese Weise können Sie bequem Nutzdaten übergeben und Statusmeldungen (Welche Schaltfläche wurde angeklickt?) an den Aufrufer melden. Hierfür sind nur sehr wenige Schritte nötig. Implementieren Sie die Activity mit Benutzeroberfläche und Programmlogik.

Denken Sie daran, der Manifestdatei ein `<activity />`-Element hinzuzufügen. Dieses erhält zusätzlich das Attribut `android:theme="@android:style/Theme.Dialog"`. Es sorgt dafür, dass das Aussehen der Activity einem Dialog entspricht. Um dem Aufrufer Ergebnisse zu übermitteln, können Sie vor dem Beenden Ihrer Activity mit `finish()` ein `Intent` erzeugen und als zweiten Parameter an `setResult()` übergeben.

5.3.3 Menüs

Menüs präsentieren dem Benutzer Funktionen bzw. Aktionen, die er zu einem bestimmten Zeitpunkt ausführen kann. Klassische Desktop-Systeme kennen neben einer Menüleiste sogenannte Kontextmenüs, die nach dem Anklicken eines Objekts mit der rechten Maustaste geöffnet werden.

Das Optionsmenü

Bis einschließlich Android 2 wurde das sogenannte *Optionsmenü* durch Drücken einer speziellen Hardwaretaste geöffnet. Seit *Honeycomb* gewährt eine virtuelle Taste in der System Bar Zugriff darauf. Es ist prinzipiell mit einer klassischen Menüleiste vergleichbar, sollte aber weitaus weniger Elemente enthalten und nur solche Funktionen anbieten, die für die aktuelle Activity sinnvoll sind.

Üblicherweise kann der Benutzer eine Einstellungsseite aufrufen oder sich Informationen über die App anzeigen lassen. Oft haben Sie auch die Möglichkeit, zur Über-

4 *http://developer.android.com/guide/topics/ui/dialogs.html*

sichtsseite einer Anwendung zurückzukehren. Wenn eine Activity ein Optionsmenü anbieten möchte, muss sie die Methode `onCreateOptionsMenu()` überschreiben. Eine typische Implementierung ist in der Klasse `MenuDemoActivity` zu sehen. Sie finden das Projekt *MenuDemo* im Verzeichnis *Quelltexte* der Begleit-DVD.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.optionsmenu, menu);
    return super.onCreateOptionsMenu(menu);
}
```

Listing 5.18 Ein Optionsmenü entfalten

Hier wird ein Menü entfaltet, dessen Elemente in der Datei *optionsmenu.xml* definiert wurden. Sie wird unter *res/menu* abgelegt und hat den folgenden Aufbau:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
          android:title="@string/item1" />
    <item android:id="@+id/item2"
          android:title="@string/item2"
          android:icon="@drawable/ic_launcher" />
</menu>
```

Listing 5.19 optionsmenu.xml

Elemente haben einen Titel. Er wird üblicherweise in *strings.xml* eingetragen und über `@string/...` referenziert. Ein Icon kann zusätzlich angegeben werden. Damit Sie auf das Anklicken eines Menüeintrags reagieren können, sollten Sie jedem Element mit `android:id` eine ID zuweisen. Die in diesem Fall auszuführenden Aktionen implementieren Sie in der Methode `onOptionsItemSelected()`.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.item1:
        tv.setText(item.getTitle());
        return true;
    case R.id.item2:
        tv.setText(item.getTitle());
        return true;
    default:
```

```
            return super.onOptionsItemSelected(item);
        }
    }
```

Listing 5.20 Auf das Anklicken von Menüelementen reagieren

Die Methode liefert `true`, wenn Ihre Activity auf das Anklicken eines Menüelements reagiert hat. Andernfalls sollten Sie die Methode der Elternklasse aufrufen und deren Ergebnis zurückliefern.

`onCreateOptionsMenu()` wird nur einmal aufgerufen. Das erzeugte Menü bleibt bis zur Zerstörung der Activity verfügbar. Wenn Sie Änderungen an Einträgen vornehmen möchten, können Sie die Methode `onPrepareOptionsMenu()` überschreiben. Android ruft sie jedes Mal vor dem Anzeigen des Optionsmenüs auf – zumindest bis einschließlich Android 2. Neuere Android-Versionen zeigen das Menü ständig an. Deshalb müssen Sie in diesem Fall selbstständig die Methode `invalidateOptionsMenu()` aufrufen.

Wenn Sie die App *MenuDemo* im Emulator starten – sie ist in Abbildung 5.12 zu sehen – fällt auf, dass auch unter Android 4 die von früheren Versionen bekannte graue Titelzeile zu sehen ist. Da virtuelle Geräte in ihrer Standardeinstellung Hardware-Tasten anbieten, ist auch kein Symbol zum Öffnen des Menüs zu sehen. Vielleicht fragen Sie sich, warum Android nicht die von »neuen« Apps bekannte *Action Bar* anzeigt. Der Grund hierfür ist der im Manifest eingetragene Wert für `android:minSdkVersion`. Eine Zahl kleiner als 11 (der API-Level von *Honeycomb*) kennzeichnet »alte« Anwendungen.

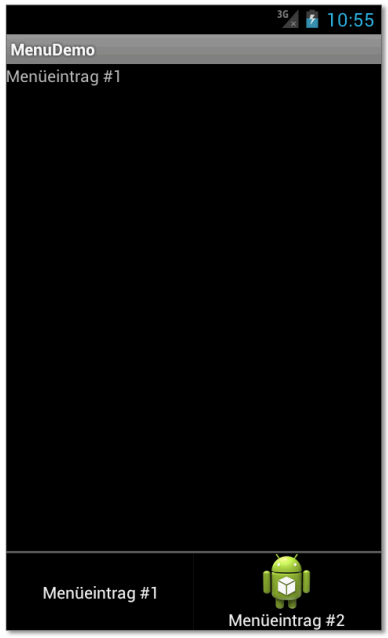


Abbildung 5.12 Die App »MenuDemo«

In Abbildung 5.13 sehen Sie die App *MenuDemo* mit geöffnetem Optionsmenü auf einem Samsung Galaxy Nexus. Dieses Gerät hat keine Hardwaretasten für die Rückkehr zum Home-Bildschirm oder zum Aufklappen des Menüs. An ihre Stelle treten die Symbole in der *System Bar* am unteren Rand des Bildschirms.

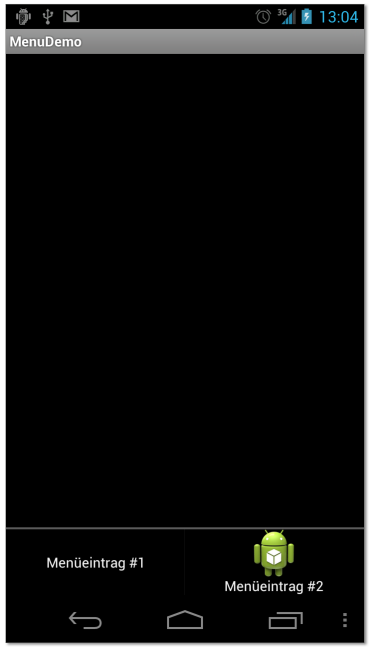


Abbildung 5.13 Ein Optionsmenü auf einem Samsung Galaxy Nexus

Wie einfach Sie bestehende Apps auf die Nutzung der Action Bar umstellen, zeige ich Ihnen gleich. Vorher möchte ich Sie aber noch mit einer weiteren Menükategorie vertraut machen.

Kontextmenüs

Android kennt das von Desktop-Systemen bekannte Konzept der Kontextmenüs. Anstelle der rechten Maustaste löst das lange Antippen (Tippen und Halten) eines Elements das Menü aus. Besonders gerne werden `ListView`s mit Kontextmenüs versehen. Aber auch andere Bedienelemente können mit solchen situationsbezogenen Menüs verknüpft werden. Nutzen Sie hierfür die Activity-Methode `registerForContextMenu()`.

Der Bau der Menüs verläuft analog zu Optionsmenüs. Sie brauchen nur die Methoden `onCreateContextMenu()` und `onContextItemSelected()` zu überschreiben und in der Ihnen bereits bekannten Weise zu implementieren. Wie dies funktioniert, ist in der Klasse `ContextMenuActivity` zu sehen. Sie ist Bestandteil des Projekts *ContextMenuDemo*, das Sie im Verzeichnis *Quelltexte* der Begleit-DVD finden.

```
public class ContextMenuDemoActivity extends Activity {

   private Button button;
   private TextView tv;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      button = (Button) findViewById(R.id.button);
      registerForContextMenu(button);
      tv = (TextView) findViewById(R.id.textview);
   }

   @Override
   public void onCreateContextMenu(ContextMenu menu,
                                   View v,
                                   ContextMenuInfo menuInfo) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.optionsmenu, menu);
   }

   @Override
   public boolean onContextItemSelected(MenuItem item) {
      switch (item.getItemId()) {
      case R.id.item1:
         tv.setText(item.getTitle());
         return true;
      case R.id.item2:
         tv.setText(item.getTitle());
         return true;
      default:
         return super.onContextItemSelected(item);
      }
   }
}
```

Listing 5.21 Die Klasse ContextMenuDemoActivity

Nach dem Start der App öffnen Sie das Kontextmenü, indem Sie die Schaltfläche Tippen und halten antippen und halten. Wenn Sie einen Menübefehl auswählen, erscheint dessen Name unterhalb des Buttons. Dies ist in Abbildung 5.14 zu sehen.

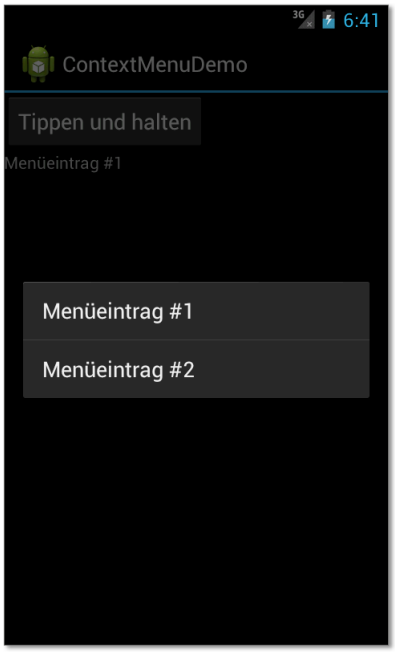


Abbildung 5.14 Die App »ContextMenuDemo«

5.3.4 Action Bar

Seit *Honeycomb* ersetzt die *Action Bar* die von älteren Android-Versionen bekannte Titelzeile am oberen Rand von Activities. Im Unterschied zu ihrem Vorgänger kann die Action Bar Funktionen übernehmen, die den Werkzeugleisten unter Windows, Mac OS X und Linux ähneln. Außerdem gewährt sie Zugang zu dem Optionsmenü, das ich Ihnen im vorangehenden Abschnitt vorgestellt habe.

Grundlegende Funktionen

Die Action Bar wird standardmäßig in allen Apps verwendet, deren Manifestdatei für mindestens eines der beiden Attribute `android:minSdkVersion` und `android:targetSdkVersion` den Wert 11 oder höher enthält. Mein Beispiel *ActionBarDemo1* zeigt, wie Sie die neuen Funktionen nutzen. Die App ist in Abbildung 5.15 zu sehen. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Bitte werfen Sie nun einen Blick auf die Klasse `ActionBarDemo1Activity`.

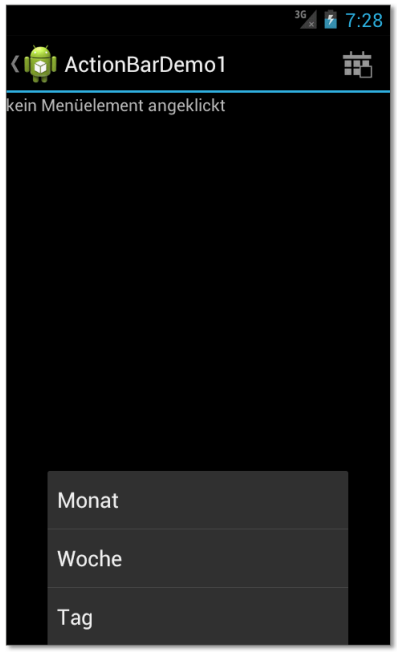


Abbildung 5.15 Die App »ActionBarDemo1«

```
public class ActionBarDemo1Activity extends Activity {

   private TextView textview;

   @Override
   protected void onStart() {
      super.onStart();
      getActionBar().setDisplayHomeAsUpEnabled(true);
   }

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      textview = (TextView) findViewById(R.id.textview);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater = getMenuInflater();
```

```
      inflater.inflate(R.menu.menu, menu);
      return super.onCreateOptionsMenu(menu);
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      textview.setText(item.getTitle());
      if (android.R.id.home == item.getItemId()) {
         Toast.makeText(this,
                        R.string.app_name,
                        Toast.LENGTH_SHORT).show();
      }
      return true;
   }
}
```

Listing 5.22 Die Klasse ActionBarDemo1Activity

Activities überschreiben wie bei Optionsmenüs die Methode `onCreateOptionsMenu()` und entfalten mit Hilfe eines `MenuInflator`s die gewünschte XML-Datei. In `onOptionsItemSelected()` reagieren Sie auf das Antippen eines Menübefehls. Meine Beispielimplementierung übernimmt den Menütitel in ein Textfeld.

Ist Ihnen aufgefallen, dass das Heute-Symbol ständig angezeigt wird? Ich habe es durch das Attribut `android:showAsAction` als sogenanntes *action item* definiert.

```
<item android:id="@+id/menu_today"
       android:icon="@android:drawable/ic_menu_today"
       android:title="@string/today"
       android:showAsAction="ifRoom|withText" />
```

Listing 5.23 Auszug aus menu.xml

Der Wert `ifRoom` legt fest, dass der Befehl in das Menü ausgelagert werden kann, wenn nicht genügend Platz im permanent sichtbaren Bereich der Action Bar zur Verfügung steht. Bei `always` unterbleibt dies. Durch das Entfernen von `withText` erscheint nur das Symbol ohne begleitende Beschriftung.

Hinweis

Die Beschriftung eines Symbols wird auch bei vorhandenem `withText` weggelassen, wenn nicht genügend Platz zur Verfügung steht. Dies ist bei meiner Beispielapp *ActionBarDemo1* im Porträtmodus der Fall. Wenn Sie den Emulator in den Quermodus bringen, erscheint die Beschriftung.

Statt eines *action items* kann die Action Bar auch Widgets enthalten. Dies ist praktisch, um beispielsweise ein Suchfeld im ständigen Zugriff zu haben. Solche *action views* werden mit dem Attribut `android:actionViewLayout` versehen. Dessen Wert referenziert eine Layout-Ressource. Alternativ kann mit `android:actionViewClass` der Klassenname des zu verwendenden Widgets festgelegt werden. Damit das Element in der Action Bar erscheint, müssen Sie das Ihnen bereits bekannte Attribut `android:showAsAction` setzen. Steht nicht genügend Platz zur Verfügung, erscheint das Element im normalen Menü. In diesem Fall verhält es sich allerdings wie ein normales Menüelement, zeigt also kein Widget an.

Aussehen und Navigation

Das in der Action Bar angezeigte Symbol entspricht normalerweise dem Standard-Icon der App. Sie können es aber durch ein Logo ersetzen. Logos dürfen breiter sein, sollten »normalen« Symbolen aber in der Höhe entsprechen. Eine solche Ressource wird dem Attribut `android:logo` der Manifestdatei zugewiesen. Ferner müssen Sie die Methode `setDisplayUseLogoEnabled()` der Action Bar mit `true` aufrufen. Activities bieten hierfür die Methode `getActionBar()` an.

Praktisch ist auch, dass Sie das Antippen des Icons mit einer Aktion versehen können. Prüfen Sie in `onOptionsItemSelected()` einfach auf `android.R.id.home == item.getItemId()`. Google empfiehlt, in diesem Fall zur Einstiegsseite der App zurückzukehren. Dies geschieht durch folgendes Quelltextfragment:

```
Intent intent = new Intent(this, StartActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
```

Tauschen Sie `StartActivity` gegen den Namen der Activity, die die Einstiegsseite Ihrer App implementiert.

Hinweis

Ab API-Level 14 müssen Sie explizit `setHomeButtonEnabled(true)` aufrufen, um auf das Antippen des Icons zu reagieren. In Honeycomb war dies noch standardmäßig so. Die im Folgenden vorgestellte Methode `setDisplayHomeAsUpEnabled()` erledigt dies für Sie.

Für Apps mit tief verschachtelten Seiten oder festgelegten Activityfolgen kann es sinnvoll sein, nicht die Einstiegsseite anzuzeigen, sondern zur nächsthöheren Ebene zurückzugehen. Die Anweisung `getActionBar().setDisplayHomeAsUpEnabled(true);` versieht das Icon bzw. Logo der Action Bar mit einem kleinen Pfeil. Sie sollten für einen solchen Aufruf die Methode `onStart()` überschreiben, weil `onCreate()` nicht zwingend bei jedem Eintritt in die Activity angesprungen wird.

Der Action Bar können noch weitere Elemente hinzugefügt werden. Beispielsweise ist es möglich, mit Hilfe einer Brotkrümelnavigation oder einer Klappliste durch Fragmente bzw. deren back stack zu blättern. Informationen hierzu finden Sie im Dokument *Action Bar*[5] der Entwicklerdokumentation. Auf der Begleit-DVD finden Sie außerdem das Projekt *TabActionBarDemo*. Es stellt, wie auch *TabActivityDemo*, eine Activity mit Registerkarten dar, nutzt hierfür aber die »neuen« Mechanismen der Action Bar.

5 *http://developer.android.com/guide/topics/ui/actionbar.html*

Kapitel 6
Multitasking

Smartphones und Tablets sind Multitalente. Während Sie im Internet surfen, können Sie nebenbei Musik hören oder sich Videoclips ansehen. In diesem Kapitel zeige ich Ihnen, wie Sie Ihre Apps fit fürs Multitasking machen.

Die Zeiten, zu denen ein Computer mehrere Programme nur *nacheinander* ausführen konnte, sind zum Glück schon sehr lange vorbei. Moderne Desktop-Systeme sind multitaskingfähig. Sie können also mehrere Anwendungen gleichzeitig ausführen. Wenn eine Maschine nur einen Mikroprozessor enthält, ist dies so natürlich eigentlich gar nicht möglich. Denn auch der Chip kann ja (normalerweise) nur ein Maschinenprogramm ausführen.

Betriebssysteme greifen deshalb zu einem Trick. Sie führen ein Programm eine gewisse Zeit lang aus und ziehen dann die Kontrolle wieder an sich. Nun kommt eine andere Anwendung an die Reihe. Dieses Spiel wiederholt sich ständig. Auch wenn also nur stets ein Programm ausgeführt wird, entsteht für den Nutzer der Eindruck einer parallelen Abarbeitung.

Auch von Betriebssystemen für Smartphones erwartet man, dass sie Multitasking unterstützen. Aber warum eigentlich? Ihr kleiner Bildschirm macht die gleichzeitige Nutzung von mehreren Anwendungen (mit Benutzeroberfläche) praktisch unmöglich. Und ein eingehender Anruf unterbricht ohnehin die aktuelle Tätigkeit. Sinnvolle Einsatzgebiete für eine parallele Abarbeitung von Aufgaben gibt es freilich viele. Denken Sie an das Abspielen von Audiotracks oder das Herunterladen von Dateien.

Android ist multitaskingfähig. Das Fundament hierfür bildet der eingesetzte Betriebssystemkern. Linux bietet sogenanntes *präemptives Multitasking* Dies bedeutet, dass Prozesse, die zu viel Rechenzeit beanspruchen, damit nicht das ganze System ausbremsen können, da nach einer gewissen Zeit der Kern die Kontrolle wieder an sich zieht. Wie Sie bereits wissen, wird jede Android-App in einer eigenen Laufzeitumgebung ausgeführt. Diese ist wiederum ein jeweils eigener Linux-Prozess. Weder ein Fehler in der App noch der Absturz ihrer virtuellen Maschine kann die Funktion des Smartphones oder Tablets also beeinträchtigen.

6.1 Threads

Auch innerhalb eines Programms kann die (quasi-)parallele Ausführung von Aufgaben sehr nützlich sein. Denken Sie an Spiele. Das Bewegen der Figur ist von Eingaben des Benutzers abhängig. Gegner oder bewegliche Hindernisse müssen aber »von alleine« ihre Position ändern können. Ein anderes Beispiel: Nehmen Sie an, das Anklicken einer Schaltfläche löst eine komplizierte Berechnung aus, die mehrere Minuten in Anspruch nimmt. Natürlich erwartet der Benutzer, dass er das Programm währenddessen weiter bedienen oder zumindest die aktuelle Tätigkeit unterbrechen kann.

6.1.1 Threads in Java

Wie Sie bereits wissen, fußt Android unter anderem auf Teilen von Apache Harmony und übernimmt damit nicht nur die Programmiersprache Java, sondern auch weite Teile der zugehörigen Klassenbibliothek. Java kennt mit sogenannten *Threads* ein Instrument zur Realisierung von *leichtgewichtigen Prozessen*. Mit solchen »Fäden« kann ein Java-Programm mehrere Tätigkeiten quasi-gleichzeitig abarbeiten.

Beteiligte Klassen und Interfaces

Die Klasse `java.lang.Thread` sowie das Interface `java.lang.Runnable` bilden die Grundlage für die Nutzung von Threads. Das folgende Beispiel zeigt, wie ein Thread erzeugt und gestartet wird. Um es auszuprobieren, legen Sie einfach ein neues Android-Projekt mit Activity an und übernehmen das Quelltextfragment. Denken Sie daran, bei der Zuweisung an `TAG` anstelle von `ThreadDemo` den Namen Ihrer Activity einzutragen.

```
private static final String TAG =
   ThreadDemo.class.getSimpleName();

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Runnable r = new Runnable() {
      @Override
      public void run() {
         Log.d(TAG, "run()-Methode wurde aufgerufen");
      }
   };
   Thread t = new Thread(r);
```

```
   t.start();
   Log.d(TAG, "Thread wurde gestartet");
}
```

Listing 6.1 Beispiel für das Starten eines Threads

Alle Anweisungen, die in einem eigenen Thread abgearbeitet werden sollen, packen Sie in die Methode `run()` einer Klasse, die das Interface `Runnable` implementiert. Diese ist übrigens nicht der Thread. Er entsteht durch den Ausdruck `Thread t = new Thread(r);`. Mit `t.start();` beginnt seine Ausführung. Sie endet, wenn alle Anweisungen innerhalb des `run()`-Methodenrumpfes abgearbeitet wurden.

Sie können den Status eines Threads übrigens mit `t.isAlive();` abfragen. Natürlich wird man für einige wenige Anweisungen, die zudem schnell abgearbeitet werden können (wie die Ausgabe in LogCat), keinen eigenen Thread starten. Sinnvoll ist sein Einsatz hingegen bei länger andauernden Berechnungen. Das folgende Quelltextfragment berechnet Fibonacci-Zahlen:

```
Runnable r = new Runnable() {
   @Override
   public void run() {
      int num = 20;
      int result = fib(num);
      Log.d(TAG, "fib(" + num + ") = " + result);
   }

   private int fib(int n) {
      switch (n) {
      case 0:
         return 0;
      case 1:
         return 1;
      default:
         Thread.yield();
         return fib(n - 1) + fib(n - 2);
      }
   }
};
```

Listing 6.2 Berechnung von Fibonacci-Zahlen in einem eigenen Thread

Die Anweisung `Thread.yield();` gibt Rechenzeit an andere parallel laufende Threads ab. Es ist übliche Praxis, dies zumindest gelegentlich zu tun. Wie lange ein Thread mit dieser Methode anhält, ist aber nicht definiert. Deshalb ist sie ungeeignet, wenn Sie

die Abarbeitung für eine bestimmte Dauer unterbrechen möchten. Denken Sie an Spiele, die Positionsänderungen von Gegnern oder beweglichen Hindernissen alle `n` Sekunden vorsehen.

Hierfür bietet die Klasse `Thread` die Methode `sleep()`.Wenn Sie das folgende Quelltextfragment in Ihre Activity übernehmen, erscheint alle drei Sekunden eine Meldung in der Sicht LogCat:

```
Runnable r = new Runnable() {
   @Override
   public void run() {
      while (true) {
         Log.d(TAG, "bewege Gegner");
         try {
            Thread.sleep(3000);
         } catch (InterruptedException e) {
            // Unterbrechungen ignorieren
         }
      }
   }
};
```

Listing 6.3 Einsatz der Methode sleep()

Ist Ihnen die Zeile `while (true) {` aufgefallen? Threads werden beendet, wenn alle Anweisungen in `run()` abgearbeitet wurden. Die Bedingung ist aber immer erfüllt. Auch der Schleifenrumpf enthält keine weiteren Abbruchgründe. Die Schleife wird also nie verlassen. Wie Sie richtig vorgehen, zeige ich Ihnen im folgenden Abschnitt.

Threads beenden

Die Klasse `Thread` beinhaltet die Methode `stop()`. Allerdings hat sich im Laufe der Zeit herausgestellt, dass deren Verwendung aus unterschiedlichen Gründen unsicher ist. Sie soll deshalb nicht verwendet werden. Technisch Interessierte finden eine ausführliche Abhandlung des Problems im Artikel *Java Thread Primitive Deprecation*. Wenn Sie von Oracle das Archiv mit der Java-Dokumentation heruntergeladen und im Installationsverzeichnis des JDK entpackt haben, finden Sie die Datei *threadPrimitiveDeprecation.html* in *docs/technotes/guides/concurrency*.

Zur Lösung des eigentlichen Problems, also des Stoppens von Threads, gibt es mehrere Ansätze. Diese haben spezifische Vor- und Nachteile. Beispielsweise ist es möglich, auf das Werfen einer `InterruptedException` mit dem Verlassen Ihrer Thread-Schleife zu reagieren. Die Methode `interrupt()` der Klasse `Thread` löst eine solche Ausnahme aus.

Sie können die Schleife stattdessen auch mit einer Abbruchbedingung versehen, die von außen gesteuert wird. In der Regel ist dies eine Instanzvariable des Typs `boolean`. Übernehmen Sie das folgende Quelltextfragment in Ihre Activity, und starten Sie anschließend die App:

```
private volatile boolean keepRunning;
private Runnable r;
private Thread t;

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Runnable-Objekt erzeugen
   r = new Runnable() {
      @Override
      public void run() {
         while (keepRunning) {
            Log.d(TAG, "bewege Gegner");
            try {
                   // 3 Sekunden warten
               Thread.sleep(3000);
            } catch (InterruptedException e) {
                   // Exception ignorieren
            }
         }
      }
   };
}

@Override
protected void onStart() {
   super.onStart();
      // Thread erzeugen
   t = new Thread(r);
   keepRunning = true;
      // Thread starten
   t.start();
}
```

Listing 6.4 Beenden eines Threads mit Abbruchbedingung

Hinweis

Das Schlüsselwort `volatile` bewirkt einen sogenannten *Cache Flush*. Dieser ist wichtig, weil das Java-Speichermodell sonst nicht gewährleistet, dass andere Threads den aktuellen Zustand der Variablen sehen. Das wiederum könnte dazu führen, dass der Thread sich niemals beendet, obwohl ein anderer die Variable `keepRunning` auf `false` gesetzt hat.

Ich habe das Erzeugen und Starten des Threads in die Methode `onStart()` ausgelagert. Sie wird nach `onCreate()` aufgerufen. Wenn Sie durch Drücken der HOME-Taste die Activity beenden, erscheinen dennoch alle drei Sekunden Meldungen in der Sicht LOGCAT. Das bloße Verlassen einer Activity führt normalerweise nicht zum Stopp von zusätzlich gestarteten Threads. Allerdings kann Android diese zum Beispiel bei Speichermangel jederzeit terminieren. Überschreiben Sie in Ihrer Activity nun die Methode `onPause()`:

```
@Override
protected void onPause() {
   super.onPause();
   keepRunning = false;
}
```

Listing 6.5 Beenden eines Threads beim Beenden einer Activity

Wenn Sie die App nun erneut starten und durch Drücken der HOME-Taste schließen, unterbleiben weitere Meldungen in der Sicht LOGCAT.

Hinweis

Das Beenden von Threads beim Verlassen einer Activity ist übrigens bewährte Praxis. Hintergrundaktivitäten werden unter Android mit sogenannten *Services* realisiert. Sie lernen diesen Grundbaustein im gleichnamigen Abschnitt 6.2 kennen. Threads können beim Bau von Services eingesetzt werden.

Threads bieten viele Möglichkeiten und geben dem Entwickler ein mächtiges Werkzeug an die Hand. Beispielsweise können Sie jedem Thread eine individuelle Priorität zuweisen und mehrere Threads zu Gruppen zusammenfassen. Allerdings erfordert insbesondere der Zugriff auf gemeinsame Ressourcen einiges an Disziplin (und die Kenntnis der Funktionsweise von `synchronized`).

Wenn Sie in Ihrer App intensiven Gebrauch von Threads machen möchten, rate ich dringend zu entsprechender Sekundärliteratur. Bitte beachten Sie hierzu die Lektüreliste im Anhang dieses Buches.

6.1.2 Vom Umgang mit Threads in Android

Wenn Sie sich die Methoden der Klasse `Thread` angesehen haben, ist Ihnen vielleicht `getName()` aufgefallen. Jedem Thread kann ein Name zugewiesen werden. Die Anweisung `Log.d(TAG, Thread.currentThread().getName());` gibt den Namen des aktuellen Threads in der Sicht LogCat aus. Sofern die App diese Anweisung nicht in einem anderen Thread ausführt, ist dies der sogenannte *Mainthread*.

Der Main- oder UI-Thread

Im *Mainthread* wird nicht nur Ihre Programmlogik ausgeführt, sondern beispielsweise auch das Zeichnen der Benutzeroberfläche. Aus diesem Grund wird er auch *UI-Thread* genannt. Welche Konsequenzen dies hat, möchte ich Ihnen anhand des Beispiels *UIThreadDemo* zeigen. Sie finden das Projekt im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

Nach dem Start der in Abbildung 6.1 gezeigten App können Sie nach Belieben Häkchen vor eine CheckBox setzen und entfernen. Der aktuelle Status (`true` oder `false`) wird unterhalb des Ankreuzfeldes angezeigt. Die Schaltfläche Berechnung starten simuliert eine länger andauernde Tätigkeit.

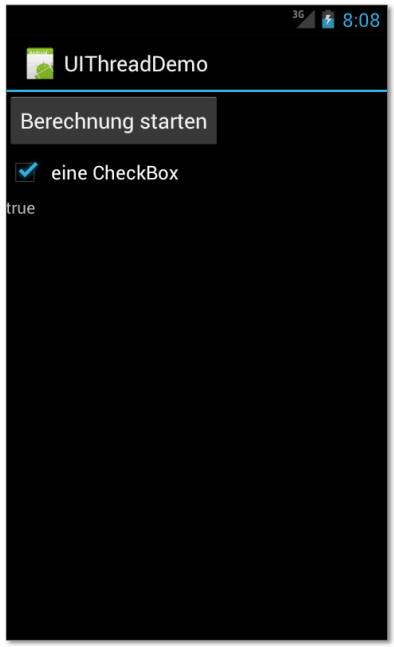


Abbildung 6.1 Die App »UIThreadDemo«

Unmittelbar nach dem Anklicken geschieht etwas Unerwartetes. Die `CheckBox` reagiert nicht mehr auf Benutzereingaben. Erst nachdem die »Berechnung« abgeschlos-

sen wurde, verhält sich die App wieder wie gewünscht. Der Grund für ihr scheinbar merkwürdiges Verhalten liegt in der Art meiner Simulation begründet. Der aktuelle Thread (also die App) wird ungefähr drei Sekunden lang schlafen geschickt.

```
try {
   Thread.sleep(3500);
} catch (InterruptedException e) {
}
```

Tauschen Sie nun das Quelltextfragment gegen die Endlosschleife

```
for (int i = 0; i < 1;);
```

aus, und starten Sie die App erneut. Anstelle der offensichtlicheren Variante `while (true)` ; habe ich die etwas verklausulierte Fassung gewählt, weil Eclipse andernfalls die Unerreichbarkeit der Anweisung `textview. setText(UIThreadDemo.this.getString(R.string.end));` moniert. Das Verhalten der App ist nahezu unverändert. Nach einer gewissen Zeit erscheint allerdings der unter Android-Entwicklern gefürchtete Dialog *Application not responding* (ANR)«. Er ist in Abbildung 6.2 zu sehen. Mit ihm informiert das System den Benutzer, dass eine Anwendung nicht mehr reagiert und bietet an, diese zu schließen oder weiter auf eine Reaktion zu warten.

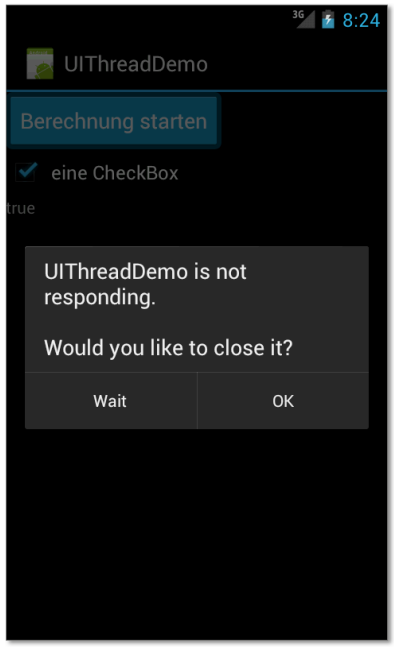


Abbildung 6.2 Frage, ob eine nicht reagierende App beendet werden soll

Wenn Sie schon Anwendungen mit Java Swing realisiert haben, ist Ihnen eine ganz ähnliche Problematik sehr wahrscheinlich vertraut. Das Pendant von Androids UI-Thread, der sogenannte *Event Dispatching Thread*, hat vielen Programmierern graue Haare beschert. Der Ruf von Swing-Anwendungen, schwerfällig zu reagieren, resultiert zu einem großen Teil aus dem nicht vorhandenen Wissen um den richtigen Einsatz von Multithreading, also der konsequenten Auslagerung von Aufgaben in Workerthreads.

Wie Java Swing ist auch der GUI-Teil von Android single-threaded. Das bedeutet: Der Mainthread ist für die Zustellung von allen Ereignissen an Widgets, aber auch für die Kommunikation Ihrer App mit den Bedienelementen zuständig. Was passiert, wenn der UI-Thread blockiert wird, können Sie mit meinem Beispiel *UIThreadDemo* sehr leicht nachvollziehen.

Um das Problem zu lösen, lagern wir die lange dauernde Berechnung in einen eigenen Thread aus. Ersetzen Sie die ursprüngliche Fassung der Methode `onClick()` durch die folgende Implementierung:

```
@Override
public void onClick(View v) {
   Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
         try {
            Thread.sleep(10000);
         } catch (InterruptedException e) {
         }
      }
   });
   t.start();
}
```

Listing 6.6 Das Anklicken der Schaltfläche startet einen neuen Thread

Testen Sie die App nun erneut. Sie verhält sich auch während der »Berechnung« normal.

Handler

Die ursprüngliche Version der App gibt zu Beginn und am Ende der Berechnung einen Text aus. Fügen Sie noch die folgenden beiden Zeilen vor bzw. nach dem `try-catch`-Block der Methode `onClick()` ein, starten Sie das Programm, und klicken Sie anschließend auf Berechnung starten.

```
textview.setText(UIThreadDemo.this.getString(R.string.begin));
textview.setText(UIThreadDemo.this.getString(R.string.end));
```

Die App stürzt ab, und Android zeigt dem Benutzer einen Hinweis, dass *UIThread-Demo* unerwartet beendet wurde. Die Ursache des Problems ist in der Sicht LogCat nachzulesen. Das System hat eine `CalledFromWrongThreadException` geworfen. Deren Nachricht lautet »Only the original thread that created a view hierarchy can touch its views.«

Bei dem »original thread« handelt es sich um den Main- bzw. UI-Thread. Es muss also eine Möglichkeit geben, Anweisungen explizit auf diesem auszuführen. Die folgende Implementierung von `onClick()` zeigt, wie Sie vorgehen:

```
@Override
public void onClick(View v) {
   final Handler h = new Handler();
   Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
         try {
            h.post(new Runnable() {
               @Override
               public void run() {
                  textview.setText(UIThreadDemo.this
                        .getString(R.string.begin));
               }
            });
            Thread.sleep(10000);
            h.post(new Runnable() {
               @Override
               public void run() {
                  textview.setText(UIThreadDemo.this
                        .getString(R.string.end));
               }
            });
         } catch (InterruptedException e) {
         }
      }
   });
   t.start();
}
```

Listing 6.7 Kommunikation mit dem UI-Thread

Als Erstes instanziieren Sie ein Objekt des Typs `android.os.Handler`. Mit ihm können Sie Nachrichten oder `Runnable`s an die Warteschlange eines Threads senden. Da beim Eintritt in `onClick()` nur der Mainthread aktiv ist, kommuniziert unser Handler mit dessen Warteschlange.

Der eigentliche Nachrichtenversand erfolgt durch Aufruf der Methode `post()`. Sie müssen also nur die auszuführende Aktion (zum Beispiel `textview.setText(UIThreadDemo.this.getString(R.string.begin));`) in eine `Runnable`-Instanz packen und an `post()` übergeben.

Android bietet einige weitere Möglichkeiten, um mit dem UI-Thread zu kommunizieren. Weiterführende Informationen zu diesem Thema finden Sie unter *Painless Threading* in der Entwicklerdokumentation.[1]

6.2 Services

Activities sind für den Benutzer unmittelbar sichtbar. Er interagiert mit ihnen. Wenn Sie eine App starten, die Musik aus dem Internet streamt, möchten Sie vielleicht zunächst ein Genre wählen und sich dann einen Sender aussuchen. Sobald die Übertragung der Daten begonnen hat, verliert die Benutzeroberfläche bei so einer App an Bedeutung. Es liegt nahe, die Activity zu verlassen, um etwas anderes zu tun – eine E-Mail zu schreiben oder im Web zu surfen.

Damit die Musik in so einem Fall nicht abbricht, muss das System eine Möglichkeit bieten, das Streamen im Hintergrund weiter auszuführen. Services sind Anwendungsbausteine, die ohne Benutzeroberfläche auskommen. Anders als beispielsweise Broadcast Receiver werden sie aber nicht nur beim Eintreten eines Ereignisses aktiviert. Auch sind Services – im Gegensatz zu den Broadcast Receivern – gerade für länger andauernde Tätigkeiten gedacht.

6.2.1 Gestartete Services

Der Bau von Activities und Broadcast Receivern folgt einem sehr ähnlichen Muster. Sie müssen Ihre Implementierungen von bestimmten Basisklassen ableiten und in der Manifestdatei der App registrieren. Auch Services entstehen auf diese Weise.

Ein einfaches Beispiel

Das Projekt *ServiceDemo1* stellt Ihnen einen sehr einfach gehaltenen Service vor. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD. Die Klasse `DemoService` lauscht auf entgangene Anrufe und protokolliert deren Anzahl in der Sicht LogCat. Sie nutzt

1 *http://developer.android.com/resources/articles/painless-threading.html*

hierfür die in Android eingebaute Anrufhistorie, das *Call Log*. Der Zugriff findet in der Methode `getMissedCalls()` statt. Ausführliche Informationen zu dieser Datenbank und ihrer Nutzung finden Sie in Kapitel 7, »Rund ums Telefonieren«.

Einfache Services leiten am besten von `android.app.Service` ab. Diese Klasse ist abstrakt. Insbesondere `onBind()` müssen Sie implementieren. Diese Methode liefert entweder `null` oder eine Instanz des Typs `android.os.IBinder`. Android unterscheidet zwischen gestarteten und gebundenen Services.

Vereinfacht ausgedrückt stellen letztere auf diese Weise Servicenutzern eine Kommunikationsschnittstelle zur Verfügung. `DemoService` tut dies nicht und liefert deshalb `null`. Die Methoden `onCreate()` und `onDestroy()` repräsentieren Stationen im Lebenszyklus eines Services und werden vom System aufgerufen. Die Beispielimplementierung registriert bzw. entfernt einen `ContentObserver`. Stellt Android Änderungen an einer Datenbank fest, wird dessen Methode `onChange()` aufgerufen.

```
public class DemoService extends Service {
   private static final String TAG =
      DemoService.class.getSimpleName();
   private ContentObserver contentObserver;

   @Override
   public IBinder onBind(Intent intent) {
      Log.d(TAG, "onBind()");
      return null;
   }

   @Override
   public void onCreate() {
      Log.d(TAG, "onCreate()");
      // ContentObserver registrieren
      contentObserver = new ContentObserver(new Handler()) {
         @Override
         public void onChange(boolean selfChange) {
            // Zahl der verpassten Anrufe ausgeben
            int missedCalls = getMissedCalls();
            Log.d(TAG, missedCalls +
                " verpasste Anrufe");
         }
      };
      getContentResolver().registerContentObserver(
            CallLog.Calls.CONTENT_URI,
            false, contentObserver);
   }
```

```
    @Override
    public void onDestroy() {
        Log.d(TAG, "onDestroy()");
        getContentResolver().unregisterContentObserver(
            contentObserver);
        contentObserver = null;
    }

    private int getMissedCalls() {
        String[] projection = { Calls._ID };
        String selection = Calls.TYPE + " = ?";
        String[] selectionArgs = {
            Integer.toString(Calls.MISSED_TYPE) };
        Cursor c =
            getContentResolver().query(CallLog.Calls.CONTENT_URI,
            projection, selection, selectionArgs, null);
        int missedCalls = c.getCount();
        c.close();
        return missedCalls;
    }
}
```

Listing 6.8 Auszug aus DemoService.java

Wie Sie bereits wissen, werden Services in der Manifestdatei der App eingetragen. Das Element `<service />` enthält mindestens das Attribut `android:name`. Ihm wird – analog zu Activities – der Name der entsprechenden Klasse zugewiesen:

```
<service android:name=".DemoService"/>
```

Weitere Attribute ermöglichen die Vergabe von Berechtigungen (`android:permission`), die Nutzung durch andere Apps (`android:exported`) sowie die Ausführung des Services in einem bestimmten Prozess (`android:process`).

Informationen hierzu finden Sie in der Beschreibung des `<service />`-Elements.[2] Damit ist die Implementierung dieses sehr einfachen Services schon abgeschlossen. Um ihn zu starten, müssen Sie innerhalb einer Activity nur die Methode `startService()` aufrufen.

```
public class ServiceDemo1 extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
```

2 *http://developer.android.com/guide/topics/manifest/service-element.html*

```
        super.onCreate(savedInstanceState);
        Intent intent = new Intent(this, DemoService.class);
        startService(intent);
    }
}
```

Listing 6.9 Auszug aus ServiceDemo1.java

Wenn Sie das Projekt *ServiceDemo1* in Ihren Eclipse-Arbeitsbereich kopiert und importiert haben, können Sie die App starten, um den Service in Aktion zu sehen (Hinweise zur Simulation von eingehenden Anrufen finden Sie in Kapitel 7, »Rund ums Telefonieren«). Versuchen Sie dies auch, nachdem Sie die Activity *ServiceDemo1* (zum Beispiel durch Drücken der HOME-Taste) verlassen haben. Nach einem verpassten Anruf erscheinen weiterhin entsprechende Ausgaben in der Sicht LOGCAT.

Wie aber erreichen Services eine solche Hintergrundverarbeitung? Da ich im ersten Abschnitt dieses Kapitels Java-Threads vorgestellt habe, liegt die Vermutung nahe, sie könnten auf dieses Hilfsmittel zurückgreifen. Dem ist aber nicht so. Services werden auf dem Mainthread ihrer App oder desjenigen Prozesses ausgeführt, der im Attribut `android:process` des `<service />`-Elements angegeben wird. Konsequenterweise müssen Sie als Entwickler rechenintensive oder lang andauernde Tätigkeiten selbstständig in eigene Threads auslagern.

Services bilden »nur« einen Rahmen, um dem System mitzuteilen, dass eine App entweder etwas im Hintergrund ausführen (selbst dann, wenn der Benutzer gar nicht mehr mit ihr interagiert) oder Teile seiner Funktionalität anderen Programmen zur Verfügung stellen möchte.

Beenden von Services

Vielleicht fragen Sie sich, wie bzw. wann Services eigentlich beendet werden. Dies ist davon abhängig, ob ein Service gestartet oder gebunden wurde (siehe folgender Abschnitt). Der Service `DemoService` wird von der Activity `ServiceDemo1` durch Aufruf der Methode `startService()` gestartet. Diese könnte ihn mit `stopService()` beenden. Services selbst stehen die Methoden `stopSelf()` und `stopSelfResult()` zur Verfügung.

Benutzer können unter EINSTELLUNGEN • APPS • AKTIV (im Emulator und auf englischsprachigen Geräten via SETTINGS • APPLICATIONS • RUNNING) durch Auswahl der App nicht mehr benötigte Services stoppen. Die Einstellungsseite ist in Abbildung 6.3 zu sehen.

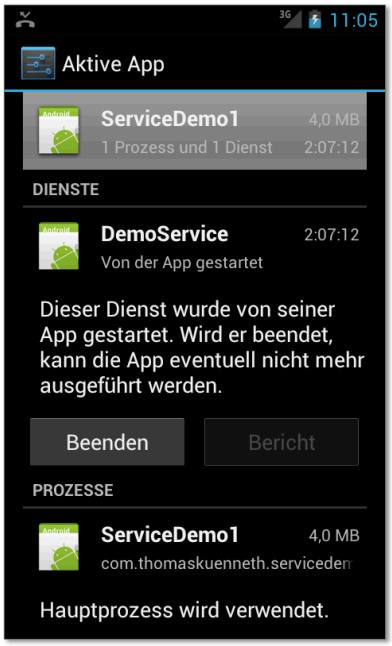


Abbildung 6.3 Seite zum Stoppen eines Services

Services, die mit `startService()` gestartet wurden, können prinzipiell unendlich lange laufen. Sie führen normalerweise genau eine Operation aus und liefern kein Ergebnis an den Aufrufer. Sie sollten sich beenden, nachdem die Aufgabe abgearbeitet wurde. Googles Dokumentation nennt das Hoch- oder Herunterladen von Dateien als Beispiel.

Nach `startService()` ruft das System die Methode `onStartCommand()` auf. Sie können diese überschreiben, um das ihr übergebene `Intent` auszuwerten. Es könnte beispielsweise den Namen der zu übertragenden Datei enthalten. Der Rückgabewert von `onStartCommand()` legt fest, wie Android verfahren soll, wenn das System den Prozess, der den Service hostet, beenden muss. Dies ist bei akutem Speichermangel der Fall.

`START_NOT_STICKY` besagt, dass der Service nur im Falle von noch ausstehenden Intents neu gestartet werden soll. Bei `START_STICKY` startet Android den Service auf jeden Fall neu und ruft `onStartCommand()` auf. Sofern keine ausstehenden Intents vorhanden sind, wird aber nicht das letzte `Intent` erneut übergeben, sondern `null`. Anders bei `START_REDELIVER_INTENT`: hier erhält `onStartCommand()` immer das letzte `Intent`. Auf diese Weise kann der Service beispielsweise die Übertragung einer Datei fortsetzen. Die Standardimplementierung liefert `START_STICKY`.

6.2.2 Gebundene Services

Die zweite von Android unterstützte Service-Art wird *gebundener Service* genannt. Solche Dienste stellen eine Schnittstelle zur Verfügung, über die Servicenutzer mit ihnen kommunizieren. Diese »Clients« können Teil der App sein, zu der auch der Service gehört. Aber auch fremde Prozesse dürfen seine Funktionen ansprechen. Ein gebundener Service läuft, solange mindestens ein Client mit ihm verbunden ist. Danach wird er zerstört.

Wie Sie bereits wissen, muss jede von `android.app.Service` abgeleitete Klasse die Methode `onBind()` bereitstellen. Sie liefert ein Objekt, das das Interface `android.os.IBinder` implementiert. `IBinder` beschreibt ein abstraktes Protokoll für die Kommunikation mit remote-fähigen Objekten. Diese wiederum bilden die Grundlage für den Aufruf von Funktionen über Prozessgrenzen hinweg, in der Informatik *Remote Procedure Call* genannt.

Die Objekte eines Java-Programms unterliegen dem Zugriff und der Kontrolle einer virtuellen Maschine. Möchte ein Objekt eine Methode eines anderen Objekts desselben Programms aufrufen, ist dies problemlos möglich. Anders sieht es aus, wenn zwei Apps miteinander kommunizieren sollen. Denn sie werden durch unterschiedliche Dalvik-Instanzen ausgeführt. Diese wiederum laufen in jeweils eigenen – streng abgeschotteten – Linux-Prozessen. Es muss also einen Mechanismus geben, der die Information, welche Funktion ausgeführt werden soll, sowie die korrespondierenden Ein- und Ausgabeparameter in geeigneter Weise transportiert.

android.os.Binder

Glücklicherweise müssen Sie das Interface `IBinder` nicht implementieren, sondern können von der Klasse `android.os.Binder` ableiten. Dies bietet sich an, wenn Ihr Service nur von der eigenen App und nur innerhalb desselben Prozesses angesprochen wird. Das Projekt *ServiceDemo2* zeigt, wie ein solcher lokaler Service aussehen kann. Kopieren Sie es aus dem Verzeichnis *Quelltexte* der Begleit-DVD in Ihren Eclipse-Arbeitsbereich und importieren es anschließend.

Die Methode `onBind()` der Klasse `LocalService` liefert eine Instanz des Typs `LocalBinder`. Dieser sieht folgendermaßen aus:

```
public class LocalBinder extends Binder {
   LocalService getService() {
      return LocalService.this;
   }
}
```

Listing 6.10 Die Klasse LocalBinder

Die Klasse leitet von `android.os.Binder` ab und enthält die zusätzliche Methode `getService()`. Diese liefert eine Referenz auf das `LocalService`-Objekt, dessen `onBind()`-Methode aufgerufen wurde. Die einzige weitere Methode von `LocalService` liefert die Fakultät der ihr übergebenen Zahl `n`.

Um den Service aufzurufen, müssen Sie ihn zunächst in der Manifestdatei registrieren. Da er die Klasse `Binder` nutzt und deshalb nur innerhalb desselben Prozesses wie der Aufrufer verwendet werden kann, setzt er das Attribut `android:exported` auf `false`. Für andere Apps ist er damit »tabu«. Die Klasse `ServiceDemo2` realisiert eine Activity mit Eingabezeile, Schaltfläche und Textfeld. Sie verbindet sich mit `LocalService` und ruft nach dem Anklicken von Berechnen dessen Methode `fakultaet()` auf.


```
public class ServiceDemo2 extends Activity {
   private LocalService mService = null;
   // wird in onStart() und onStop() verwendet
   private ServiceConnection mConnection =
      new ServiceConnection() {
         @Override
         public void onServiceConnected(ComponentName name,
            IBinder service) {
               LocalBinder binder = (LocalBinder) service;
               mService = binder.getService();
      }

      @Override
      public void onServiceDisconnected(ComponentName name) {
         mService = null;
      }
   };

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      final TextView textview =
         (TextView) findViewById(R.id.textview);
      final EditText edittext =
         (EditText) findViewById(R.id.edittext);
      final Button button =
         (Button) findViewById(R.id.button);
      button.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            if (mService != null) {
```

```
                // Service aufrufen und
                // Fakultät berechnen lassen
                int n = Integer.parseInt(
                   edittext.getText().toString());
                int fak = mService.fakultaet(n);
                textview.setText(getString(
                   R.string.template, n, fak));
             }
          }
       });
    }

    @Override
    protected void onStart() {
       super.onStart();
       Intent intent = new Intent(this, LocalService.class);
       bindService(intent, mConnection,
          Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
       super.onStop();
       if (mService != null) {
          unbindService(mConnection);
          mService = null;
       }
    }
}
```

Listing 6.11 Auszug aus ServiceDemo2

Um die Verbindung zu einem Service herzustellen, müssen Sie die Methode `bindService()` aufrufen. Dies geschieht in `onStart()`. Um `ServiceDemo2` von `LocalService` zu trennen, nutze ich `unbindService()`. In beiden Fällen wird die Referenz auf ein `ServiceConnection`-Objekt übergeben.

Dessen Callback-Methode `onServiceConnected()` weist den übergebenen `IBinder` (nach einem Cast) der `ServiceDemo2`-Instanzvariable `mService` zu. Hierbei handelt es sich um den Rückgabewert von `onBind()` aus `LocalService`. Damit ist der Aufruf von `fakultaet()` möglich:

```
int n = Integer.parseInt(edittext.getText().toString());
int fak = mService.fakultaet(n);
```

android.os.Messenger

Die Kommunikation mit einem Service über Prozessgrenzen hinweg kann auf zweierlei Weise erfolgen. Die meisten Freiheiten bietet die Nutzung von *AIDL*, der *Android Interface Definition Language*. Allerdings rät Google unter anderem aus Komplexitätsgründen von der direkten Nutzung von AIDL für den Bau von Services ab. Trotzdem möchte ich Ihnen ein paar grundlegende Informationen darüber geben:

Objekte werden in primitive Einheiten zerlegt und vom Betriebssystem an den Zielprozess übermittelt. Um AIDL zu nutzen, müssen Sie die gewünschte Kommunikationsschnittstelle in einer *.aidl*-Datei ablegen. Die Werkzeuge des Android SDK erzeugen daraus eine abstrakte Klasse. Sie implementiert die von Ihnen definierten Methoden und kümmert sich um die Interprozesskommunikation. Weiterführende Informationen finden Sie im Dokument *Android Interface Definition Language (AIDL)*.[3]

Auch die Klasse `android.os.Messenger` funktioniert über Prozessgrenzen hinweg. Die Idee ist, in einem Prozess einen `Messenger` zu instanziieren. Dieser referenziert einen `Handler`, dem Nachrichten übermittelt werden können. Das `Messenger`-Objekt wird dann einem anderen Prozess übergeben.

Um zu demonstrieren, wie Sie dieses zugegebenermaßen nicht ganz leicht verständliche Konzept praktisch umsetzen können, habe ich die beiden Projekte *ServiceDemo3_Service* und *ServiceDemo3* im Verzeichnis *Quelltexte* der Begleit-DVD abgelegt. Sie implementieren den eigentlichen Service sowie eine Nutzer-App. Diese ist ohne den Service nicht lauffähig. Kopieren Sie die beiden Projekte in Ihren Eclipse-Arbeitsbereich, und importieren Sie sie anschließend.

`com.thomaskuenneth.servicedemo3_service.RemoteService` ist recht einfach aufgebaut. Die Klasse leitet von `android.app.Service` ab. Außer einer privaten Methode zur Berechnung der Fakultät implementiert sie nur `onBind()`. Die hier gelieferte Referenz auf eine `IBinder`-Instanz entsteht, indem die Methode `getBinder()` eines `Messenger`-Objekts aufgerufen wird. Dieses wiederum kommt folgendermaßen zustande:

```
private final Messenger mMessenger =
   new Messenger(new IncomingHandler());
```

Die Klasse `IncomingHandler` enthält die eigentliche Kommunikation. Die Implementierung überschreibt die Methode `handleMessage()`. Ihr wird ein `Message`-Objekt übergeben, das eine eingehende Nachricht repräsentiert. Enthält dessen Instanzvariable `what` einen bestimmten Wert, wird die Fakultät der Zahl aus `arg1` berechnet. Das

3 *http://developer.android.com/guide/developing/tools/aidl.html*

Ergebnis wird in Gestalt eines eigenen, neuen `Message`-Objekts an den Absender der gerade bearbeiteten Nachricht (`Messenger m = msg.replyTo;`) übertragen.

```
private class IncomingHandler extends Handler {
   @Override
   public void handleMessage(Message msg) {
      switch (msg.what) {
      case MSG_FAKULTAET_IN:
         Integer n = msg.arg1;
         Log.d(TAG, "Eingabe: " + n);
         int fak = fakultaet(n);
         Messenger m = msg.replyTo;
         Message msg2 = Message.obtain(null,
            MSG_FAKULTAET_OUT, n, fak);
         try {
            m.send(msg2);
         } catch (RemoteException e) {
            Log.e(TAG, "send()", e);
         }
         break;
      default:
         super.handleMessage(msg);
      }
   }
}
```

Listing 6.12 Die Klasse IncomingHandler

Die Berechnung der Fakultät wird also in zwei Mitteilungen aufgeteilt. Die beiden `int`-Werte `MSG_FAKULTAET_IN` (Berechnung starten) und `MSG_FAKULTAET_OUT` (Ergebnis zurückliefern) müssen auch dem Servicenutzer bekannt sein. Um einen Service aus einer fremden App heraus nutzen zu können, müssen Sie ihn exportieren. Warum Sie zudem einen Intent-Filter brauchen, erkläre ich Ihnen gleich.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.thomaskuenneth.servicedemo3_service"
      android:versionCode="1"
      android:versionName="1.0">
   <uses-sdk android:minSdkVersion="9" />
   <application android:icon="@drawable/icon"
         android:label="@string/app_name">
    <service android:name=".RemoteService"
         android:exported="true">
```

```
        <intent-filter>
           <action
               android:name="com.thomaskuenneth.servicedemo3_service.
               RemoteService" />
        </intent-filter>
     </service>
  </application>
</manifest>
```

Listing 6.13 Manifestdatei von ServiceDemo3_Service

Das Projekt *ServiceDemo3* nutzt den `RemoteService` aus *ServiceDemo3_Service*. Seine Oberfläche entspricht *ServiceDemo2*. In der Methode `onStart()` wird der Service gebunden. Dies geschieht durch einen Aufruf von `bindService()`. Wie Sie bereits wissen, wird hierfür ein `Intent` benötigt.

Anders als bei lokalen Services können Sie dieses aber nicht durch `new Intent(this, RemoteService.class)` instanziieren. Bedenken Sie, dass die App die Klasse `RemoteService` gar nicht kennt. Aus diesem Grund übergebe ich eine Aktion in Form der Zeichenkette `com.thomaskuenneth.servicedemo3_service.RemoteService`. Sie entspricht dem Wert des Attributs `android:name` des `<action />`-Elements in der Manifestdatei von *ServiceDemo3_Service*. Auf diese Weise kann Android den Empfänger des Intents ermitteln.

`bindService()` erhält als zweiten Parameter die Referenz auf ein `ServiceConnection`-Objekt.

```
private ServiceConnection mConnection = new ServiceConnection() {
   public void onServiceConnected(ComponentName className,
      IBinder service) {
         mService = new Messenger(service);
   }
   public void onServiceDisconnected(ComponentName className) {
      mService = null;
   }
};
```

Listing 6.14 Auszug aus ServiceDemo3.java

Die Variable `mService` verweist letztlich also auf den Service aus *ServiceDemo3_Service*. Damit er die Fakultät einer ihm übergebenen Zahl berechnet, sind nur wenige Zeilen Code nötig. Zunächst wird mit `Message.obtain()` eine neue Nachricht erzeugt und mit `mService.send()` verschickt:

```
if (mService != null) {
   int n = Integer.parseInt(edittext.getText().toString());
   Message msg = Message.obtain(null, MSG_FAKULTAET_IN, n, 0);
   msg.replyTo = mMessenger;
   try {
      mService.send(msg);
   } catch (RemoteException e) {
      Log.d(TAG, "send()", e);
   }
}
```

Listing 6.15 Auszug aus ServiceDemo3.java

Eine Kleinigkeit bleibt noch zu tun. Wie Sie wissen, möchte `RemoteService` das Ergebnis seiner Berechnung als Nachricht versenden. Zur Erinnerung nochmals das zuständige Quelltextfragment:

```
Messenger m = msg.replyTo;
Message msg2 = Message.obtain(null,
   MSG_FAKULTAET_OUT, n, fak);
try {
   m.send(msg2);
} catch (RemoteException e) {
   Log.e(TAG, "send()", e);
}
```

Listing 6.16 Auszug aus RemoteService.java

In *ServiceDemo3.java* habe ich `msg.replyTo` auf `mMessenger` gesetzt. Diese Variable referenziert eine `Messenger`-Instanz. Sie enthält, wie Sie bereits wissen, einen `Handler`. Wie üblich wird dessen Methode `handleMessage()` überschrieben. Sie schreibt die Werte aus `arg1` (Zahl, deren Fakultät berechnet werden sollte) und `arg2` (das Ergebnis) in ein Textfeld. Eine solche Ausgabe ist in Abbildung 6.4 zu sehen.

```
private final Messenger mMessenger = new Messenger(
   new IncomingHandler());
private class IncomingHandler extends Handler {
   @Override
   public void handleMessage(Message msg) {
      switch (msg.what) {
      case MSG_FAKULTAET_OUT:
         int n = msg.arg1;
         int fakultaet = msg.arg2;
         Log.d(TAG, "Fakultaet: " + fakultaet);
```

```
            textview.setText(getString(R.string.template,
               n, fakultaet));
            break;
         default:
            super.handleMessage(msg);
         }
      }
}
```

Listing 6.17 Auszug aus ServiceDemo3.java

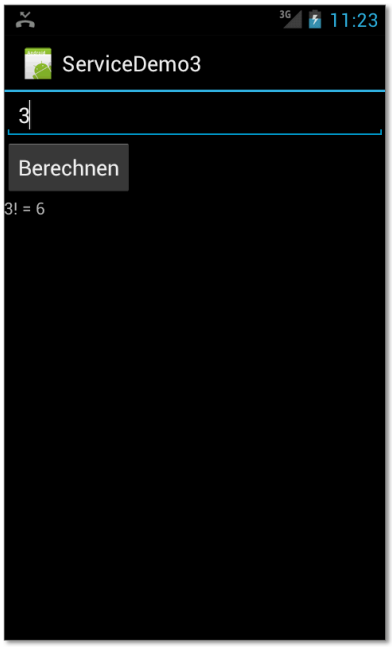


Abbildung 6.4 Die App »ServiceDemo3«

Die Erstellung von remotefähigen Services ist sicher eine nicht ganz einfache Aufgabe. Insofern sollten Sie sehr genau prüfen, ob Sie den Aufwand wirklich betreiben müssen oder ob auch ein lokaler Service ausreichend ist.

TEIL III

Telefonfunktionen nutzen

Kapitel 7
Rund ums Telefonieren

In diesem Kapitel zeige ich Ihnen, wie Sie Anrufe in Apps tätigen, entgegennehmen und wie Sie den Netzstatus ermitteln. Außerdem lernen Sie das Call Log kennen, sozusagen das Gedächtnis der Android-Telefoniefunktion.

Vielleicht fragen Sie sich, warum Sie in Ihren Apps Anrufe tätigen sollten, obwohl das Android-System eine bedienfreundliche Wählfunktion beinhaltet? Dasselbe gilt für das *Call Log*, also die Historie getätigter, empfangener und entgangener Anrufe. Diese Informationen sind auf der Registerkarte ANRUFE der App *Telefon* zu sehen.

Die Kunst der Entwicklung mobiler Anwendungen besteht darin, Vorhandenes in neuem Kontext wiederzuverwenden und so für den Benutzer einen echten Mehrwert zu schaffen.

Stellen Sie sich eine Aufgabenverwaltung vor. Sie haben als »To do« eingetragen, einen Termin mit Ihrem Steuerberater zu vereinbaren. Wenn Sie die Aufgabe antippen, bietet Ihnen die App an, eine E-Mail zu erstellen oder einen Anruf zu tätigen. Die technischen Grundlagen sind durch die Ihnen bereits bekannten Intents vorhanden. Ihnen als App-Entwickler kommt die keinesfalls immer einfache Aufgabe zu, diese Bausteine auf innovative, sinnvolle Weise zu verbinden.

7.1 Telefonieren

Um ein Telefonat zu beginnen, sind nur sehr wenige Zeilen Quelltext nötig. Auch das Entgegennehmen von Anrufen verursacht kaum Aufwand.

7.1.1 Anrufe tätigen

Am einfachsten initiieren Sie ein Gespräch, indem Sie die zu wählende Nummer an die eingebaute, in Abbildung 7.1 dargestellte Telefon-Anwendung übergeben.

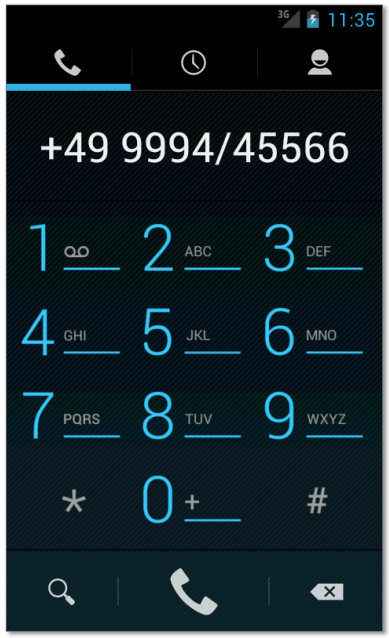


Abbildung 7.1 Der Wähldialog von Android 4

Da der Benutzer sieht, dass er einen Anruf tätigen wird, sind hierfür keine speziellen Berechtigungen erforderlich. Der Aufruf funktioniert so:

```
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:"
      + "+49 (999) 44 55 66"));
startActivity(intent);
```

Wie Sie bereits aus Kapitel 4, »Activities und Broadcast Receiver«, wissen, ist die Methode `startActivity()` in allen Klassen vorhanden, die von `android.content.Context` ableiten, also beispielsweise in Activities und Services.

Das Projekt *AnrufDemo* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches demonstriert, wie Sie das Quelltextfragment in eine Activity einbetten. Die App stellt hierzu die beiden Schaltflächen WÄHLDIALOG und SOFORT WÄHLEN dar. Klicken Sie auf letztere, beginnt das Programm sofort zu wählen. Um dies zu erreichen, verwenden Sie statt `ACTION_DIAL` `ACTION_CALL`:

```
Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:"
      + "+49 (999) 44 55 66"));
startActivity(intent);
```

Wichtig ist hierbei aber, dass Sie in der Manifestdatei das Recht `android.permission.CALL_PHONE` eintragen. Fehlt dies, wird die App beim Wählversuch mit einer Fehlermeldung beendet.

7.1.2 Auf eingehende Anrufe reagieren

Auch das Reagieren auf eingehende Anrufe ist mit Android sehr einfach möglich. Wie Sie hierzu vorgehen, zeige ich Ihnen nun. Importieren Sie *TelephonyManagerDemo* in Ihren Eclipse-Arbeitsbereich, und sehen Sie sich die Klasse `TelephonyManagerDemo` an. Sie finden das Projekt im Verzeichnis *Quelltexte* der Begleit-DVD.

In der Methode `onCreate()` wird zunächst mit `getSystemService(TELEPHONY_SERVICE)` die Referenz auf ein Objekt des Typs `android.telephony.TelephonyManager` ermittelt.

Sie verwenden dieses Objekt, um durch Aufruf der Methode `listen()` einen sogenannten `PhoneStateListener` zu registrieren. Seine Methoden werden aufgerufen, wenn bestimmte telefoniebezogene Ereignisse eintreten. Worüber Android den Listener informieren soll, regelt der zweite Parameter der Methode `listen()`, beispielsweise `LISTEN_CALL_STATE`.

```
public void onCallStateChanged(int state, String incomingNumber) {
   StringBuilder sb = new StringBuilder();
   sb.append("Status: " + state + "\n");
   sb.append("Eingehende Rufnummer: " + incomingNumber + "\n");
   textview.setText(sb.toString());
}
```

Listing 7.1 Auf eingehende Anrufe reagieren

Die Beispielimplementierung der für diesen Fall zu überschreibenden Methode `onCallStateChanged()` fügt ihre Parameter zu einer Zeichenkette zusammen und gibt diese in einer TextView aus. Die Variable `state` gibt Auskunft darüber, aus welchem Grund die Methode aufgerufen wurde:

- `CALL_STATE_RINGING` signalisiert einen eingehenden Anruf.
- `CALL_STATE_OFFHOOK` wird gemeldet, wenn der Benutzer das Gespräch angenommen hat.
- `CALL_STATE_IDLE` schließlich kennzeichnet das Ende eines Gesprächs.

Um die Benachrichtigung bei Statuswechseln des Telefons zu beenden, rufen Sie die Methode `listen()` mit `LISTEN_NONE` auf. Hierzu ein Beispiel:

```
@Override
protected void onDestroy() {
   super.onDestroy();
   mgr.listen(psl, PhoneStateListener.LISTEN_NONE);
}
```

Listing 7.2 Auszug aus TelephonyManagerDemo.java

Um den Status des Telefons auslesen zu können, ist die Berechtigung `android.permission.READ_PHONE_STATE` nötig.

Vielleicht fragen Sie sich nun, wie Sie eingehende Anrufe simulieren können. Hierzu können Sie eine Telnet-Verbindung zum Emulator herstellen. Sehen Sie zunächst in der Titelzeile des Emulator-Fensters nach dem verwendeten Port. In der Regel ist dies 5554.

Öffnen Sie anschließend eine *Shell* bzw. *Eingabeaufforderung*, und geben Sie dort `telnet localhost 5554` (bzw. die bei Ihnen angezeigte Nummer) ein. Unter Windows 7 müssen Sie u. U. erst den `telnet`-Client aktivieren (entsprechende Hinweise finden Sie im Microsoft-Supportdokument *Aktivierung des Telnet-Clients unter Windows 7*)[1]. Wenn die `telnet`-Verbindung zum Emulator steht, setzen Sie das Kommando `gsm call <Nummer>` ab.

Die *Android Development Tools* beinhalten die Eclipse-Perspektive *DDMS* (*Dalvik Debug Monitor Server*). Ihre Sichten und Editoren bieten unter anderem Zugriff auf das Dateisystem, den Heap sowie Threads von echten Geräten und dem Emulator.

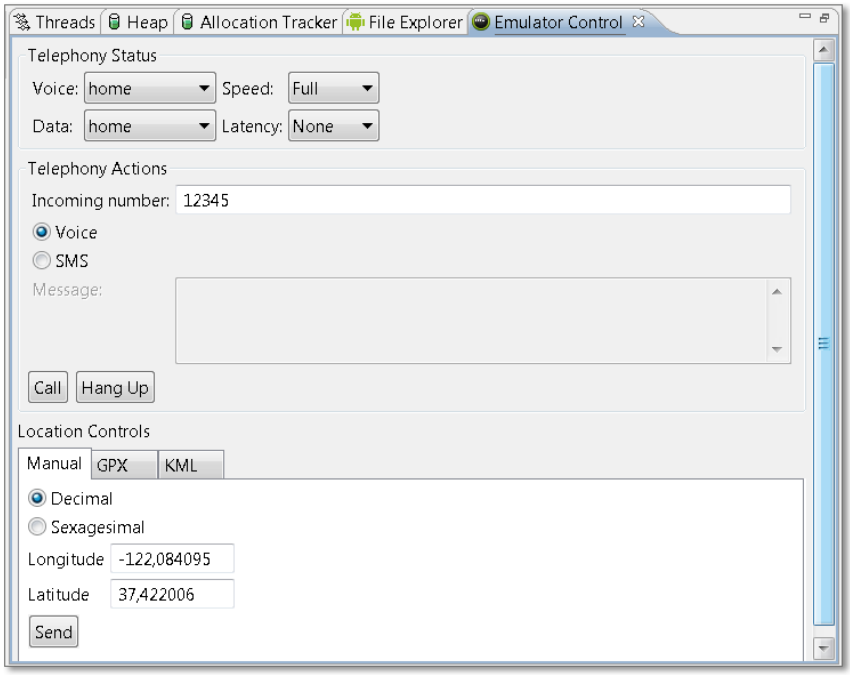


Abbildung 7.2 Die Sicht »Emulator Control«

1 *http://support.microsoft.com/kb/978779/de*

Mit der in Abbildung 7.2 gezeigten Sicht Emulator Control können Sie ebenfalls Anrufe simulieren. Dies ist unter anderem praktisch, um das Verhalten des im folgenden Abschnitt angesprochenen Call Logs zu studieren.

7.2 Telefon- und Netzstatus

Android-Geräte haben am liebsten immer Zugang zu Netzen. Die wichtigsten Informationen, beispielsweise Art der Verbindung oder die Signalstärke, werden deshalb permanent in der Statusleiste angezeigt. Selbstverständlich lassen sie sich bequem durch eigene Apps ermitteln.

7.2.1 Geräte identifizieren

Grundsätzlich sollten Sie anstreben, mit Ihren Apps so wenig benutzerbezogene Daten zu verarbeiten und zu speichern wie möglich. Insbesondere das eindeutige Identifizieren eines Geräts über dessen 15-stellige IMEI-Nummer (das Kürzel bedeutet *International Mobile Equipment Identity*) müssen Sie Ihren Anwendern unbedingt mitteilen, um nicht in den Verdacht zu geraten, heimlich Daten zu sammeln. Natürlich kann es gute Gründe dafür geben, eine solche ID an einen Server zu übermitteln. Denken Sie an Sicherheitsfunktionen wie das ferngesteuerte Löschen des Smartphones nach einem Verlust oder Diebstahl.

Die im vorherigen Abschnitt angesprochene Klasse `TelephonyManager` enthält die Methode `getDeviceId()`. Sie liefert die im GSM-Netz verwendete IMEI oder aber MEID bzw. ESN als Pendant in CDMA-Netzen. Beachten Sie, dass Sie im Manifest die Berechtigung `android.permission.READ_PHONE_STATE` anfordern müssen.

```
TelephonyManager mgr = (TelephonyManager)
     getSystemService(Context.TELEPHONY_SERVICE);
Log.d(TAG, mgr.getDeviceId());
```

Über die Klasse `Settings.Secure` ist der lesende Zugriff auf zahlreiche Systemeinstellungen möglich. Sie enthält unter anderem `ANDROID_ID`, eine als Hex-String-kodierte 64-Bit-Zufallszahl. Sie wird beim erstmaligen Start des Systems erzeugt und später nicht mehr verändert (außer unter Umständen beim Zurücksetzen des Geräts auf die Werkseinstellungen).

```
Log.d(TAG, Settings.Secure.getString(getContentResolver(),
      Settings.Secure.ANDROID_ID));
```

Um `ANDROID_ID` auszulesen, sind keine besonderen Berechtigungen erforderlich.

7.2.2 Netzwerkinformationen anzeigen

Das folgende Quelltextfragment liest Basisinformationen über die aktuell vorhandenen Netzwerke aus. Um es einsetzen zu können, müssen Sie in der Manifestdatei die Berechtigung `android.permission.ACCESS_NETWORK_STATE` anfordern.

```
ConnectivityManager mgr = (ConnectivityManager)
     getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo[] networks = mgr.getAllNetworkInfo();
for (NetworkInfo n : networks) {
   Log.d(TAG,
         n.getTypeName() + " ("
               + n.getSubtypeName() + ")");
   Log.d(TAG, "isAvailable(): " + n.isAvailable());
   Log.d(TAG, "isConnected(): " + n.isConnected());
   Log.d(TAG, "roaming ist "
         + (n.isRoaming() ? "ein" : "aus"));
}
```

Listing 7.3 Informationen über die verfügbaren Netzwerke ausgeben

Wenn Sie Informationen zu einem bestimmten Netzwerktyp benötigen, können Sie anstelle von `getAllNetworkInfo()` die Methode `getNetworkInfo()` aufrufen. Die hierbei zu übergebenden Typ-Konstanten sind in der Klasse `ConnectivityManager` enthalten, beispielsweise `TYPE_WIFI`.

`isAvailable()` liefert `true`, wenn die Verbindung zu einem Netzwerk grundsätzlich möglich ist. Allerdings können bestimmte Bedingungen einen Connect verhindern, zum Beispiel wenn sich der Benutzer nicht in seinem Heimatnetz befindet (dies kann mit `isRoaming()` abgefragt werden) und Roaming ausgeschaltet hat.

7.3 Das Call Log

Das *Call Log* speichert Informationen über getätigte und empfangene Anrufe. Es ist über die Registerkarte ANRUFE der App *Telefon* erreichbar. Der Zugriff auf seine Daten ist für die verschiedensten Anwendungsfälle interessant. Statistik-Apps könnten beispielsweise das Anrufverhalten visualisieren. Mit einem Live Wallpaper ließen sich die am häufigsten kontaktierten Freunde und Bekannte als Tag Cloud darstellen. Oder denken Sie an ein Widget, das über entgangene Anrufe informiert. In Kapitel 8, »Widgets und Wallpapers«, greife ich diese Idee wieder auf.

7.3.1 Entgangene Anrufe

Um auf das Call Log zuzugreifen, brauchen Sie keinen `PhoneStateListener` zu registrieren. Stattdessen können Sie einfach eine Datenbankabfrage auf `CallLog.Calls.CONTENT_URI` absetzen. Wie das funktioniert, zeigt die folgende Methode `getMissedCalls()`. Damit sie funktioniert, muss die App in der Manifestdatei die Berechtigung `android.permission.READ_CONTACTS` anfordern.

```
public List<CallLogData> getMissedCalls(ContentResolver r) {
   List<CallLogData> l = new ArrayList<CallLogData>();
   String[] projection = { Calls.NUMBER, Calls.DATE,
      Calls.NEW, Calls._ID };
   String selection = Calls.TYPE + " = ?";
   String[] selectionArgs = {
      Integer.toString(Calls.MISSED_TYPE) };
   Cursor c = r.query(CallLog.Calls.CONTENT_URI,
      projection, selection,
      selectionArgs, null);
   while (c.moveToNext()) {
      String number = c.getString(0);
      long date = c.getLong(1);
      int isNew = c.getInt(2);
      long id = c.getLong(3);
      l.add(new CallLogData(number, date, isNew, id));
   }
   c.close();
   return l;
}
```

Listing 7.4 Entgangene Anrufe ermitteln

Die Methode liefert eine Liste der entgangenen Anrufe. Diese werden in `CallLogData`-Objekten abgelegt. Die Klasse ist sehr einfach aufgebaut und verzichtet bewusst auf die in Java gerne verwendeten Getter und Setter. Bedenken Sie, dass auf Smartphones trotz ihrer erstaunlichen Leistungsfähigkeit Speicher noch immer eine knappe Ressource ist. `CallLogData` hält in der hier gezeigten Form nicht alle möglichen Elemente vor. Bei Bedarf könnten Sie die Klasse zum Beispiel um die Anrufdauer in Sekunden (`Calls.DURATION`) erweitern.

```
public class CallLogData {
   public final String number;
   public final long date;
   public final boolean isNew;
   public final long id;
```

```
   public CallLogData(String number, long date,
      int isNew, long id) {
         this.number = number;
         this.date = date;
         this.isNew = (isNew != 0);
         this.id = id;
   }
}
```

Listing 7.5 Sammler für Daten aus dem Call Log

`getMissedCalls()` benötigt die Referenz auf einen `ContentResolver`, wie sie beispielsweise die Methode `getContentResolver()` liefert. Sie ist in allen von `android.content.Context` abgeleiteten Klassen vorhanden.

Tipp

Das Datum eines Anrufs wird im Call Log in Millisekunden seit dem 1. Januar 1970 GMT gespeichert. Um komfortabel damit rechnen zu können, sollten Sie es deshalb mit `new java.util.Date(...)` umwandeln.

7.3.2 Einträge bearbeiten

Die Daten des Call Logs können durch Apps nicht nur gelesen, sondern auch verändert werden. Sie können sich dies beispielsweise zunutze machen, um den Status `Calls.NEW` auf `false` zu setzen. Damit werden Anrufe nicht mehr als »neu« angezeigt. Dies funktioniert folgendermaßen:

```
// Liste entgangener Anrufe ermitteln
List<CallLogData> l = getMissedCalls(getContentResolver());
for (CallLogData d : l) {
   Log.d(TAG, d.number + (d.isNew ? " (neu)" : ""));
   // Hatte der Anrufer die Telefonnummer 123?
   if ("123".equals(d.number) && d.isNew) {
      ContentValues values = new ContentValues();
      values.put(Calls.NEW, 0);
      String where = Calls._ID + " = ?";
      String[] selectionArgs = { Long.toString(d.id) };
      int numRows = getContentResolver().
            update(Calls.CONTENT_URI,
            values, where, selectionArgs);
      Log.d(TAG, numRows + " rows updated");
   }
```

```
}

l = getMissedCalls(getContentResolver());
for (CallLogData d : l) {
   Log.d(TAG, d.number + (d.isNew ? " (neu)" : ""));
}
```

Listing 7.6 Das Call Log bearbeiten

Bauen Sie dieses Quelltextfragment in eine Activity ein, und simulieren Sie vor dem Start zwei Anrufe, wobei einer von der Telefonnummer 123 ausgehen muss. Nehmen Sie die Gespräche im Emulator nicht an, sondern beenden Sie beide in der Sicht Emulator Control, indem Sie die Schaltfläche Hang Up anklicken. Da durch die Update-Anweisung Daten verändert werden, müssen Sie im Manifest die Berechtigung `android.permission.WRITE_CONTACTS` anfordern.

7.3.3 Benachrichtigung bei Änderungen

Ich habe Ihnen gezeigt, wie Sie entgangene Anrufe ermitteln können. Damit ein Widget deren Zahl anzeigen kann, muss es nach einer Änderung informiert werden. Ein regelmäßiger Aufruf der Methode `getMissedCalls()` (Informatiker nennen das *Polling*) verbraucht aber unnötig Rechenzeit und damit Energie. Natürlich bietet Android hierfür eine elegante Lösung. `ContentResolver` bieten die Möglichkeit, sich bei Datenbankänderungen informieren zu lassen. Um dies auszuprobieren, legen Sie die Klasse `MyService` an, die von `android.app.Service` ableitet. Sie implementiert also einen Service. Überschreiben Sie die Methode `onCreate()`, und fügen Sie die folgenden Zeilen in den Methodenrumpf ein:

```
contentObserver = new ContentObserver(new Handler()) {
   @Override
   public void onChange(boolean selfChange) {
      Log.d(TAG, "onChange()");
   }
};
getContentResolver().registerContentObserver(
   CallLog.Calls.CONTENT_URI,
   false, contentObserver);
```

Damit dies nicht zu Fehlern führt, benötigen Sie noch folgende Instanz- und Klassenvariablen:

```
private static final String TAG = MyService.class.getSimpleName();
private ContentObserver contentObserver;
private final IBinder mBinder = new LocalBinder();
```

Bei jeder Änderung am Call Log wird die Methode `onChange()` aufgerufen. Anstelle der simplen Log-Ausgabe könnten Sie beispielsweise die Anzahl der entgangenen Anrufe ermitteln. Fügen Sie Ihrem Service zur Vollendung noch das folgende Quelltextfragment hinzu:

```
@Override
public void onDestroy() {
   Log.d(TAG, "onDestroy()");
   getContentResolver().unregisterContentObserver(contentObserver);
   contentObserver = null;
}

@Override
public IBinder onBind(Intent intent) {
   Log.d(TAG, "onBind()");
   return mBinder;
}

public class LocalBinder extends Binder {
   MyService getService() {
      return MyService.this;
   }
}
```

Listing 7.7 Auszug aus MyService.java

Interessant ist vor allem die Methode `onDestroy()`, die beim Beenden eines Services aufgerufen wird. Sie entfernt den in `onCreate()` registrierten `ContentObserver`.

Services werden in der Manifestdatei registriert. Dies geschieht mit folgendem Eintrag, der sich innerhalb des `application`-Elements befinden muss:

```
<service android:name="MyService" />
```

Der letzte Schritt ist das Starten des Services, zum Beispiel innerhalb einer Activity. Dies könnte so aussehen:

```
startService(new Intent(this, MyService.class));
```

Das Call Log ist ein wichtiges Instrument bei der Nutzung eines Mobiltelefons. Insofern sollten Sie es in Ihren Apps mit Bedacht einsetzen und den Anwender bei Änderungen informieren.

Kapitel 8
Widgets und Wallpapers

Mit Widgets und Wallpapers können Sie den Startbildschirm Ihres Android-Geräts individuell gestalten. Wie Sie solche Komponenten selbst entwickeln, zeige ich Ihnen in diesem Kapitel.

Der Startbildschirm ist die Steuerzentrale aller Android-Geräte. Er bietet Zugriff auf den Programmstarter, mit dem Sie sowohl mitgelieferte als auch nachträglich installierte Apps öffnen. Außerdem können Sie Systemeinstellungen ändern und Programme verwalten. Der Hintergrund dieser »Home«-Anwendung zeigt entweder ein frei änderbares Bild (*Wallpaper*) oder beliebig komplexe Animationen. Solche *Live Wallpapers* haben mit Android 2.1 (API-Level 7) Einzug in das System gehalten.

Außerdem können Sie *Widgets* (ich verwende gelegentlich das Synonym »Appwidget«) auf dem Hintergrund ablegen. Hierbei handelt es sich um kleine, in ihrer Größe veränderbare Info- oder Steuertäfelchen, die neben dem aktuellen Wetter, dem Füllstand des Akkus oder der Signalstärke der Drahtlosnetzwerke zum Beispiel Bedienelemente für Hintergrundprozesse enthalten. Als Benutzer können Sie auf diese Weise ein anderes Lied oder Album auswählen, ohne in die Mediaplayer-App wechseln zu müssen.

8.1 Widgets

Entwickler können schon seit der Android-Version 1.5 (Cupcake) Widgets für den Home-Bildschirm erstellen. Entsprechend riesig ist die Auswahl in Google Play. Dennoch möchte ich Sie ermutigen, Ihre eigenen Infotäfelchen zu programmieren. Denn Appwidgets nutzen zahlreiche Konzepte, die ich in den vorangehenden Kapiteln vorgestellt habe. Sie sind also eine prima Spielwiese, um das Erlernte auszuprobieren. Außerdem können Sie in der Benutzeroberfläche Ihre Kreativität ausleben.

8.1.1 Beteiligte Klassen und Dateien

Appwidgets sind schnell geschrieben. Es handelt sich im Prinzip um normale Apps, die aber bestimmte Klassen implementieren und bestimmte Dateien enthalten müssen. Welche dies sind, demonstriere ich Ihnen anhand der App *Entgangene Anrufe*. Sie fin-

den das korrespondierende Projekt *Missed Calls Widget* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Das in Abbildung 8.1 dargestellte Widget zeigt die Anzahl der Anrufe an, die von der Gegenseite beendet wurden, bevor Sie das Gespräch angenommen haben. Wird es angetippt, öffnet sich das *Call Log*. Ausführliche Informationen zur Anrufhistorie von Android finden Sie in Kapitel 7, »Rund ums Telefonieren«. Dort zeige ich Ihnen auch, wie Sie Telefonate im Emulator simulieren können.

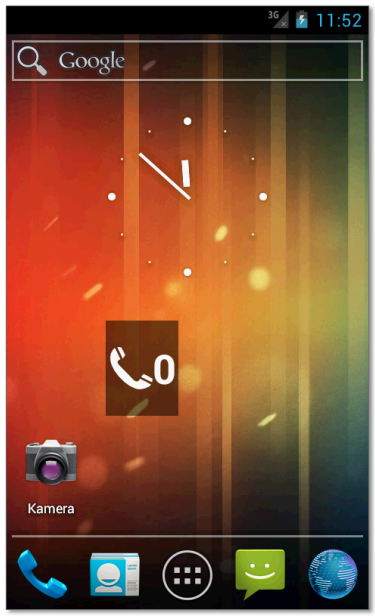


Abbildung 8.1 Das Widget »Entgangene Anrufe«

Einträge in der Manifestdatei

Die Basisklasse für die Implementierung Ihrer Widgets ist üblicherweise `android.appwidget.AppWidgetProvider`. Diese wiederum erbt von `android.content.BroadcastReceiver`. Android kommuniziert also mit Appwidgets, indem das System ihnen Intents sendet. Diese werden durch die `onReceive()`-Implementierung von `AppWidgetProvider` ausgewertet. Sie müssen nur wenige Methoden überschreiben, um auf die Ereignisse zu reagieren. Da Widgets indirekt von `BroadcastReceiver` ableiten, fügen Sie der Manifestdatei entsprechende `<receiver />`-Elemente hinzu.

Das Attribut `android:name` verweist wie gewohnt auf die implementierende Klasse. Ein Intent-Filter sorgt dafür, dass Aktionen vom Typ `android.appwidget.action.APPWIDGET_UPDATE` an den Receiver übermittelt werden. `com.thomaskuenneth.missedcalls.MISSED_CALLS` ist projektspezifisch, kommt also nur im *Missed Calls Widget* vor. Diese Aktion wird von der Klasse `MissedCallsService` verschickt. Wie Sie bereits wissen, werden Services im Tag `<service />` der Manifestdatei eingetragen. Weitere Informationen zu der Klasse `MissedCallsService` finden Sie im folgenden Abschnitt.

Die beiden Attribute des Elements `<meta-data />` müssen bei jeder `<receiver />`-Definition für Widgets angegeben werden. Der Wert von `android:name` bleibt stets gleich und lautet `android.appwidget.provider`. `android:resource` hingegen verweist auf eine unter *res/xml* abgelegte Konfigurationsdatei. Sie enthält zusätzliche Informationen zu einem Widget. Da diese in erster Linie die Benutzeroberfläche betreffen, stelle ich sie Ihnen im entsprechenden Abschnitt ausführlicher vor.

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

<application android:icon="@drawable/icon"
  android:label="@string/app_name">

  <receiver android:name=".MissedCallsWidget"
    android:label="@string/app_name">
    <intent-filter>
      <action
      android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <intent-filter>
      <action
    android:name="com.thomaskuenneth.missedcalls.MISSED_CALLS" />
    </intent-filter>
    <meta-data android:name="android.appwidget.provider"
               android:resource="@xml/missedcallswidget_info" />
  </receiver>

  <service android:name="MissedCallsService" />
```

Listing 8.1 Auszug aus der Manifestdatei von Missed Calls Widget

MissedCallsService

Die Klasse `MissedCallsService` stellt einen Service zur Verfügung, der durch das Appwidget `MissedCallsWidget` mit `startService()` gestartet und mit `stopService()` beendet wird. Seine statische Methode `getMissedCalls()` liefert die Anzahl der entgangenen Anrufe. Diese lassen sich sehr bequem der Ihnen bereits bekannten Anrufhistorie entnehmen. `MissedCallsService` registriert einen `ContentObserver`, der bei Änderungen an der Anrufhistorie ein Broadcast Intent feuert.

```
public class MissedCallsService extends Service {

   public static final String ACTION_MISSED_CALL =
      "com.thomaskuenneth.missedcalls.MISSED_CALLS";

   public static final String EXTRA_MISSED_CALL =
      "MISSED_CALLS";

   private static final String TAG =
      MissedCallsService.class.getSimpleName();

   private ContentObserver contentObserver;

   @Override
   public IBinder onBind(Intent intent) {
      Log.d(TAG, "onBind()");
      // Service wird nicht mit bindService() gebunden
      return null;
   }

   @Override
   public void onCreate() {
      Log.d(TAG, "onCreate()");
      contentObserver = new ContentObserver(
         new Handler()) {
            @Override
            public void onChange(boolean selfChange) {
               int missedCalls =
                  getMissedCalls(MissedCallsService.this);
            Log.d(TAG, "onChange() - " +
               missedCalls +
               " entgangene Anrufe");
            // Intent erstellen und senden
            Intent i =
               new Intent(ACTION_MISSED_CALL);
            i.putExtra(EXTRA_MISSED_CALL,
               missedCalls);
            sendBroadcast(i);
         }
      };
      getContentResolver().registerContentObserver(
         CallLog.Calls.CONTENT_URI,
         false, contentObserver);
```

```
    }

    @Override
    public void onDestroy() {
        Log.d(TAG, "onDestroy()");
        getContentResolver().unregisterContentObserver(
            contentObserver);
        contentObserver = null;
    }

    public static int getMissedCalls(Context context) {
        String[] projection = { Calls._ID };
        String selection = Calls.TYPE + " = ?";
        String[] selectionArgs = {
            Integer.toString(Calls.MISSED_TYPE) };
        Cursor c = context.getContentResolver().query(
            CallLog.Calls.CONTENT_URI,
            projection, selection,
            selectionArgs, null);
        int missedCalls = c.getCount();
        c.close();
        Log.d(TAG, "getMissedCalls() - " +
            missedCalls + " entgangene Anrufe");
        return missedCalls;
    }
}
```

Listing 8.2 Auszug aus MissedCallsService.java

Wird der Service nicht mehr benötigt, muss der `ContentObserver` entfernt werden. Dies geschieht in der Methode `onDestroy()`.

Hinweis

Android beendet Services bei Speichermangel automatisch. Auch Anwender haben die Möglichkeit, sie zu stoppen. In solchen Fällen kann das Widget natürlich nicht mehr über neue entgangene Anrufe informieren. Sie können aber `MissedCallsService` zum Beispiel durch Aufruf der Methode `startForeground()` vor dem Entfernen bei knappem Speicher schützen.

MissedCallsWidget

Die Klasse `MissedCallsWidget` repräsentiert das eigentliche Widget und leitet deshalb von `AppWidgetProvider` ab. Sie können die Methoden `onEnabled()` und `onDisabled()`

der Elternklasse überschreiben, um beim Aktivieren der ersten bzw. Deaktivieren der letzten Widget-Instanz (Anwender können Appwidgets mehrfach auf dem Home-Bildschirm ablegen) bestimmte Tätigkeiten auszuführen. In meiner Implementierung starte bzw. beende ich den Service `MissedCallsService`. Auch das Anlegen oder Löschen von temporären Datenbanken könnte dort stattfinden.

```
@Override
public void onEnabled(Context context) {
  Log.d(TAG, "onEnabled()");
  super.onEnabled(context);
  Intent service = new Intent(context, MissedCallsService.class);
  context.startService(service);
}

@Override
public void onDisabled(Context context) {
  Log.d(TAG, "onDisabled()");
  super.onDisabled(context);
  Intent service = new Intent(context, MissedCallsService.class);
  context.stopService(service);
}
```

Listing 8.3 Auszug aus MissedCallsWidget.java

Da `AppWidgetProvider` eingehende Intents schon an Callback-Methoden wie `onEnabled()` oder `onDisabled()` delegiert, müssen Sie in Ihren Widgets `onReceive()` meistens nicht überschreiben. `MissedCallsWidget` tut dies, weil der Service `MissedCallsService` nach Änderungen an der Anrufhistorie die Aktion `ACTION_MISSED_CALL` als Broadcast Intent verschickt. Auf dieses muss das Widget aber reagieren, um die übertragene Anzahl entgangener Anrufe anzeigen zu können. Aus diesem Grund wurde im `<receiver />`-Element der Manifestdatei ein entsprechender Intent-Filter definiert.

```
@Override
public void onReceive(Context context, Intent intent) {
   Log.d(TAG, "onReceive()");
   super.onReceive(context, intent);
   if (intent != null) {
      if (MissedCallsService.ACTION_MISSED_CALL
            .equals(intent.getAction())) {
               int missedCalls = intent.getIntExtra(
                  MissedCallsService.EXTRA_MISSED_CALL, -1);
         if (missedCalls != -1) {
            ...
```

```
            }
        }
    }
}
```

Listing 8.4 Auszug aus MissedCallsWidget.java

Wenn die Benutzeroberfläche eines Widgets aktualisiert werden muss, wird die Methode `onUpdate()` aufgerufen. Ausführliche Informationen zu ihr finden Sie im folgenden Abschnitt. Ich habe die eigentliche Implementierung in die private Methode `updateWidgets()` ausgelagert und mit einem zusätzlichen Parameter `missedCalls` versehen. Wird `updateWidgets()` nämlich aus `onUpdate()` heraus aufgerufen, muss das Widget die Zahl der entgangenen Anrufe vorher mithilfe der Methode `getMissedCalls()` aus `MissedCallsService` ermitteln.


```
@Override
public void onUpdate(Context context,
    AppWidgetManager appWidgetManager,
    int[] appWidgetIds) {
        updateWidgets(context,
        appWidgetManager, appWidgetIds,
        MissedCallsService.getMissedCalls(context));
}
```

Listing 8.5 Das Widget aktualisieren

Im Rahmen der Verarbeitung von `MissedCallsService.ACTION_MISSED_CALL` ist dies hingegen nicht nötig, weil das Intent die Anzahl der entgangenen Anrufe schon mitbringt. Allerdings müssen alle Instanzen eines Widgets »von Hand« aktualisiert werden, weil die Basisklasse `AppWidgetProvider` die von mir definierte Aktion `MissedCallsService.ACTION_MISSED_CALL` natürlich nicht kennt. Das folgende Quelltextfragment zeigt, wie Sie hierbei vorgehen. `context` verweist auf eine Instanz des Typs `android.content.Context`.

```
AppWidgetManager m = AppWidgetManager.getInstance(context);
if (m != null) {
    int[] appWidgetIds =
        m.getAppWidgetIds(new ComponentName(context,
        getClass()));
    if ((appWidgetIds != null) && (appWidgetIds.length > 0)) {
        updateWidgets(context, m, appWidgetIds, missedCalls);
    }
}
```

Listing 8.6 Auszug aus MissedCallsWidget.java

`android.appwidget.AppWidgetManager` liefert Informationen über installierte Appwidget-Provider und ermöglicht Statusänderungen von Widgets. Um darauf zuzugreifen, müssen Sie eine Instanz dieser Klasse ermitteln. Dies geschieht mit `AppWidgetManager.getInstance()`. Anschließend erhalten Sie durch Aufruf der Methode `getAppWidgetIds()` alle Instanzen eines Widgets. Da ich die Aktualisierung der Benutzeroberfläche in eine eigene Methode ausgelagert habe, können wir diese bequem mit `updateWidgets()` anspringen.

8.1.2 Die Benutzeroberfläche

Die Benutzeroberfläche von Widgets wird wie gewohnt in Layoutdateien definiert und zur Laufzeit zu einem Objektbaum entfaltet. Allerdings stehen nicht alle Views und ViewGroups zur Verfügung. Eingesetzt werden können zum Beispiel `FrameLayout`, `LinearLayout` und `RelativeLayout` sowie `AnalogClock`, `Button`, `Chronometer`, `ImageButton`, `ImageView`, `ProgressBar` und `TextView`.

`MissedCallsWidget` entfaltet die folgende Layoutdatei *missedcallswidget_layout.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/datewidget_id"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="@dimen/widget_margin">

   <RelativeLayout
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:background="#80000000"
   >

      <ImageView
          android:id="@+id/missedcallswidget_image"
         android:src="@drawable/telefon"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_alignParentLeft="true"
         android:layout_marginLeft="4dp"
         android:layout_centerVertical="true"
         />

       <TextView
          android:id="@+id/missedcallswidget_count"
```

```
            android:layout_alignParentRight="true"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:layout_marginRight="2dp"
             android:layout_centerVertical="true"
             style="@style/Text.VeryBig" />

    </RelativeLayout>

</FrameLayout>
```

Listing 8.7 missedcallswidget_layout.xml

Die Layoutdatei enthält nur wenige Elemente. Sicher ist Ihnen aufgefallen, dass ein `<RelativeLayout />` ein `<ImageView />` sowie ein `<TextView />` positioniert. Vielleicht fragen Sie sich aber, warum ich das Ganze noch in ein `<FrameLayout />` verpacke, das zudem ein Attribut mit merkwürdigem Inhalt hat:

```
android:padding="@dimen/widget_margin"
```

Für frühere Android-Versionen sahen Googles Richtlinien vor, dass Widgets an allen Seiten freie Bereiche haben mussten. Seit Android 4 fügt die Plattform diesen selbst hinzu, Widgets müssen sich darum also nicht mehr kümmern. Würde ein Appwidget ihn dennoch setzen, wäre der ihm zur Verfügung stehende Platz geringer als nötig. Damit Ihre App unter allen Android-Versionen läuft, wird der freie Bereich nicht fest in der Layoutdatei eingetragen, sondern als versionsabhängige Ressource in der Datei *dimens.xml* definiert.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="AppWidget_1Cell">72dp</dimen>
    <dimen name="widget_margin">8dp</dimen>
</resources>
```

Listing 8.8 Die Datei dimens.xml in res/values

Die Dimension `widget_margin` hat den Wert `8dp`. Wird dieser in einer Layoutdatei verwendet, entsteht um das Widget ein jeweils 8 *density-independent pixel* (dp) großer Bereich. Dieses Verhalten soll auf älteren Android-Versionen greifen. Aus diesem Grund wird die Datei unter *res/values* abgelegt.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="AppWidget_1Cell">40dp</dimen>
```

```
   <dimen name="widget_margin">0dp</dimen>
</resources>
```

Listing 8.9 Die Datei dimens.xml in res/values-v14

Für Android-Versionen ab 4.0 darf ein Widget keinen eigenen freien Bereich mehr belegen. Deshalb wird in der Datei *dimens.xml* für `widget_margin` der Wert `0dp` eingetragen. Diese Version wird unter *res/values-v14* abgelegt.

onUpdate()

Wie Sie bereits wissen, wird die Methode `onUpdate()` aufgerufen, wenn die Benutzeroberfläche eines Widgets aktualisiert werden muss. Dies geschieht das erste Mal nach dem Hinzufügen zum Home-Bildschirm. Eine Ausnahme bilden Appwidgets mit Einstellungsseite. In solchen Fällen unterbleibt der Aufruf von `onUpdate()`, und die Konfigurationsactivity muss den ersten Aufruf selbst herbeiführen. Wie dies funktioniert, habe ich Ihnen im vorherigen Abschnitt gezeigt. Alle weiteren Aufrufe von `onUpdate()` erfolgen nach einer von Ihnen festzulegenden Zeitspanne. Weitere Informationen hierzu finden Sie im folgenden Abschnitt.

Wenn ein Widget Ereignisse verarbeitet, die durch den Benutzer ausgelöst werden, müssen diese innerhalb von `onUpdate()` registriert werden. Dies kann zum Beispiel durch Aufruf der Methode `setOnClickPendingIntent()` geschehen.

```
private void updateWidgets(Context context,
   AppWidgetManager appWidgetManager, int[] appWidgetIds,
   int missedCalls) {
      // Layoutdatei entfalten
      final RemoteViews updateViews = new RemoteViews(

         context.getPackageName(),
            R.layout.missedcallswidget_layout);
      // Zahl entgangener Anrufe ausgeben
      updateViews.setTextViewText(R.id.missedcallswidget_count,
         Integer.toString(missedCalls));
      // auf Antippen reagieren
      Intent intent = new Intent(Intent.ACTION_VIEW,
         CallLog.Calls.CONTENT_URI);
      intent.setType(CallLog.Calls.CONTENT_TYPE);
      PendingIntent pendingIntent =
         PendingIntent.getActivity(context, 0, intent, 0);
      updateViews.setOnClickPendingIntent(R.id.datewidget_id,
         pendingIntent);
```

```
        appWidgetManager.updateAppWidget(appWidgetIds,
            updateViews);
}
```

Listing 8.10 Auszug aus MissedCallsWidget.java

Um die Benutzeroberfläche aus einer Layoutdatei zu entfalten, instanziieren Sie ein Objekt des Typs `RemoteViews`. Der Zugriff auf Attribute einer View erfolgt dann über dieses Objekt. Beispielsweise setzen Sie den Text einer `TextView` mit `setTextView-Text()`. Richtlinien zur Gestaltung von Widgets hat Google im Dokument *App Widget Design Guidelines* zusammengefasst.[1]

Größe von Widgets und initiales Layout

In der Konfigurationsdatei eines Widgets tragen Sie dessen minimale Größe ein und legen das initiale Layout fest. Die Datei wird unter *res/xml* abgelegt.

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
     xmlns:android="http://schemas.android.com/apk/res/android"
   android:minWidth="@dimen/AppWidget_1Cell"
   android:minHeight="@dimen/AppWidget_1Cell"
   android:updatePeriodMillis="1800000"
   android:initialLayout="@layout/widget_initial_layout"
   android:previewImage="@drawable/preview"
/>
```

Listing 8.11 Die Datei missedcallswidget_info.xml

Missed Calls Widget verwendet beispielsweise die Datei *missedcallswidget_info.xml*. Das Pflichtattribut `android:initialLayout` verweist auf ein Layout, das so lange angezeigt wird, bis es beispielsweise in `onUpdate()` ersetzt wird.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
     xmlns:android="http://schemas.android.com/apk/res/android"
   android:text="@string/loading_data"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   style="@style/Text.Big" />
```

Listing 8.12 Die Datei widget_initial_layout.xml

1 *http://developer.android.com/guide/practices/ui_guidelines/widget_design.html*

Die Werte `android:minWidth` und `android:minHeight` in der Datei *missedcallswidget_info.xml* legen fest, wie viel Platz ein Widget auf dem Home-Bildschirm mindestens einnehmen möchte. Sie werden in *density-independent pixels* angegeben. Die insgesamt zur Verfügung stehende Fläche wird in eine bestimmte Anzahl von Zeilen und Spalten unterteilt. Jede so entstandene Zelle hat eine festgelegte Größe. Entsprechend seiner minimalen Breite und Höhe belegt ein Widget also eine bestimmte Anzahl von Zellen.

Bei der Konzeption Ihres Widgets überlegen Sie sich, wie viele solcher Zellen es breit und hoch sein soll. Beispielsweise sind 1 × 1 und 4 × 1 übliche Größen. Um die daraus resultierenden Werte für `android:minWidth` und `android:minHeight` zu berechnen, wurde früher die Formel `(Anzahl der Zellen * 74) – 2` verwendet. Eine Zelle war demzufolge `72dp` breit oder hoch, vier Zellen entsprachen `294dp`. Auch dies hat Google mit Android 4 geändert. Die neue Formel lautet `(70 * Anzahl der Zellen) – 30`. Um diese Versionsabhängigkeit aufzulösen, verwenden wir wie schon bei dem leeren Bereich an den Rändern eines Appwidgets die Ihnen bereits bekannte Datei *dimens.xml*.

Das Attribut `android:updatePeriodMillis` gibt an, wie oft Android von einem `AppWidgetProvider` eine Aktualisierung anfordern soll. Hierzu wird, wie Sie bereits wissen, die Methode `onUpdate()` aufgerufen. Generell sollte ein Appwidget so selten wie möglich aktualisiert werden. Die Häufigkeit hängt natürlich von der Art des Widgets ab. Werte unter 30 Minuten werden vom System ignoriert.

Das optionale Attribut `android:configure` verweist auf eine Activity, die gestartet wird, wenn der Benutzer dem Home-Bildschirm ein Widget hinzufügt. Sie sollte eine Einstellungsseite anzeigen. Ausführliche Informationen hierzu finden Sie im Dokument *App Widgets*.[2]

8.1.3 Vorschau-Bilder

Mit dem optionalen Attribut `android:previewImage` können Sie eine Grafik festlegen, die Androids Widget-Auswahl anstelle eines Icons anzeigt. In meinem Beispiel bedeutet `@drawable/preview`, dass hierfür die Datei *preview.png* verwendet werden soll. Wie schon bei den Programm-Icons landet die Standard-Version der Grafik in *res/drawable*. Versionen für unterschiedliche Pixeldichten können Sie zusätzlich in den Ihnen bereits bekannten Verzeichnissen (beispielsweise *drawable-hdpi* oder *drawable-ldpi*) ablegen.

Für das Erstellen von Vorschau-Grafiken steht in Android 4-basierten virtuellen Geräten die App *Widget Preview* zur Verfügung. Sie ist in Abbildung 8.2 zu sehen. Nach dem Auswählen des Appwidgets, für das Sie eine Vorschaugrafik erstellen möchten, legen Sie die Datei durch Anklicken von Take Snapshot im Verzeichnis

2 *http://developer.android.com/guide/topics/appwidgets/index.html*

Download des externen Speichermediums ab. Von dort aus kopieren Sie es in der Sicht *File Explorer* mit PULL A FILE FROM THE DEVICE auf Ihren Entwicklungsrechner.

Abbildung 8.2 Die App Widget Preview

Die Auswahl eines Widgets mit Vorschaugrafik ist in Abbildung 8.3 zu sehen.

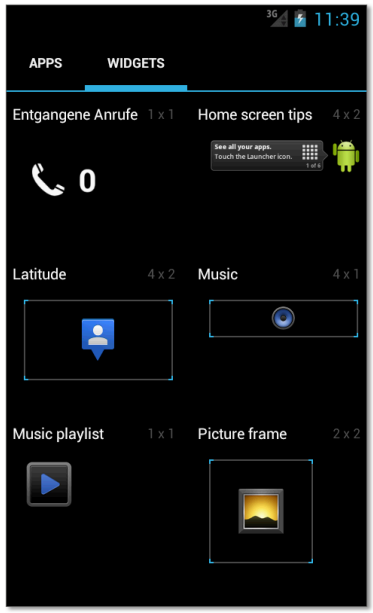


Abbildung 8.3 Die Appwidget-Auswahl unter Android 4

8.2 Wallpaper

Android-Anwender können ihr System durch statische und animierte Hintergründe individualisieren. Das Austauschen dieser Grafiken ist auch durch eigene Apps möglich. Wie Sie hierzu vorgehen, möchte ich Ihnen im Folgenden zeigen.

Legen Sie ein neues Projekt an, und lassen Sie den Assistenten eine Activity namens `WallpaperDemo` generieren. Setzen Sie hierzu ein Häkchen vor CREATE ACTIVITY, und geben Sie den Namen im dafür vorgesehenen Feld ein. Fügen Sie am Ende des Methodenrumpfes von `onCreate()` die Zeile

```
WallpaperManager m = WallpaperManager.getInstance(this);
```

hinzu. `this` bezieht sich auf die aktuelle Activity. Sie leitet indirekt von `android.content.Context` ab. Dies ist auch bei Services so. Das Ändern des Hintergrunds ist also auch aus solchen Anwendungsbausteinen heraus möglich. In Broadcast Receivern können Sie innerhalb der Methode `onReceive()` Wallpapers setzen. Damit lässt sich beispielsweise nach jedem Start des Systems ein Wechsel des Motivs herbeiführen. Ihr Receiver muss hierzu auf `Intent.ACTION_BOOT_COMPLETED` reagieren.

8.2.1 Die Wallpaper-API

Nachdem Sie mit `getInstance()` die Referenz auf ein `WallpaperManager`-Objekt ermittelt haben, können Sie mit `setResource()` eine Grafik aus *res/drawable* als neuen Hintergrund setzen.

```
try {
   m.setResource(R.drawable.icon);
} catch (IOException e) {
   Log.e(TAG, "Fehler bei setResource()", e);
}
```

Damit das funktioniert, müssen Sie in der Manifestdatei der App die Berechtigung `android.permission.SET_WALLPAPER` eintragen.

Die Grafik sollte zu der Größe und Pixeldichte des aktuellen Geräts passen. Sie erreichen dies, indem Sie entsprechend angepasste Dateien in den Verzeichnissen *drawable-hdpi*, *drawable-ldpi*, *drawable-mdpi* bzw. *drawable-xhdpi* ablegen. Programmatisch können Sie die gewünschte Breite und Höhe in Pixeln mithilfe der beiden Methoden `getDesiredMinimumWidth()` und `getDesiredMinimumHeight()` ermitteln:

```
Log.d(TAG, "gewünschte Mindestgröße: " +
      m.getDesiredMinimumWidth() +
      "x" +
      m.getDesiredMinimumHeight() + " Pixel");
```

Das ist praktisch, wenn Sie keine fertige Datei laden, sondern eine eigene `Bitmap` zur Laufzeit der App zusammenstellen möchten. Das folgende Quelltextfragment demonstriert dies und zeichnet zwei sich kreuzende blaue Linien auf weißen Grund. Zusätzlich wird in schwarzer Schrift der Text »Hallo Android!« ausgegeben. Der Aufruf `setBitmap()` übernimmt die Grafik als Wallpaper.

```
int w = m.getDesiredMinimumWidth();
int h = m.getDesiredMinimumHeight();
Bitmap bm = Bitmap.createBitmap(w, h, Config.RGB_565);
Canvas c = new Canvas(bm);
Paint paint = new Paint();
paint.setTextAlign(Align.CENTER);
paint.setColor(Color.WHITE);
c.drawRect(0, 0, w - 1, h - 1, paint);
paint.setColor(Color.BLUE);
c.drawLine(0, 0, w - 1, h - 1, paint);
c.drawLine(0, h - 1, w - 1, 0, paint);
paint.setColor(Color.BLACK);
c.drawText("Hallo Android!", w / 2, h / 2, paint);
try {
   m.setBitmap(bm);
} catch (IOException e) {
   Log.e(TAG, "Fehler bei setBitmap()", e);
}
```

Listing 8.13 Beispiel für eine einfache Grafikausgabe

Wie Sie sehen, ist es mit wenig Aufwand möglich, einfache Grafiken mit der Klasse `android.graphics.Canvas` zu zeichnen. Weiterführende Informationen finden Sie im Dokument *Graphics*.[3]

8.2.2 Hintergründe auswählen

Um dem Anwender die Möglichkeit zu geben, einen Hintergrund auszuwählen, instanziieren Sie ein Intent mit der Action `ACTION_SET_WALLPAPER` und übergeben es an `startActivity()`.

```
Intent i = new Intent(Intent.ACTION_SET_WALLPAPER);
startActivity(i);
```

Hierdurch wird der in Abbildung 8.4 gezeigte Auswahldialog von Android gestartet. Mit ihm bestimmt der Benutzer die App zum Festlegen eines neuen Hintergrunds.

3 *http://developer.android.com/guide/topics/graphics/index.html*

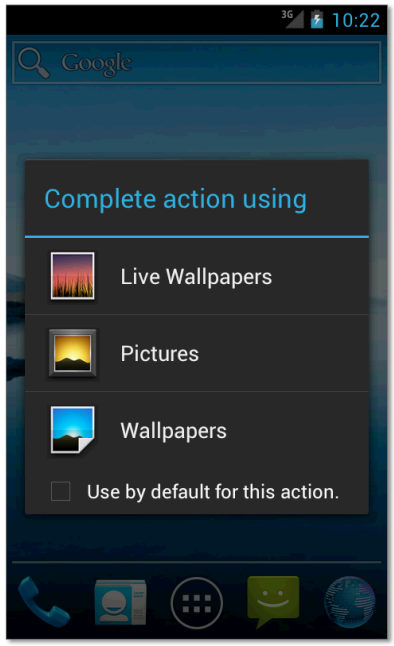


Abbildung 8.4 Dialog zum Auswählen eines Hintergrunds

Allerdings erscheint dieser Dialog nur, sofern noch keine Standard-Activity für das gewünschte Intent festgelegt wurde. In diesem Fall wird es nämlich sofort an diese bevorzugte Aktivität übermittelt. Sie können das Festlegen einer Standard-Activity durch den Benutzer aber verhindern. Kapseln Sie hierzu das eigentliche Intent (zum Beispiel `Intent.ACTION_SET_WALLPAPER`) in ein `ACTION_CHOOSER`-Intent.

Sie können auf diese Weise auch eine Überschrift für den in Abbildung 8.5 gezeigten Dialog festlegen. Das korrespondierende Quelltextfragment sieht folgendermaßen aus:

```
Intent target = new Intent(Intent.ACTION_SET_WALLPAPER);
startActivity(Intent.createChooser(target,
   "Hintergrund auswählen mit..."));
```

Vielleicht fragen Sie sich, warum Sie den Benutzer daran hindern sollten, für bestimmte Intents eine Standard-Activity festzulegen. Es gibt viele Situationen, in denen der Anwender gerne mit seiner Lieblings-App arbeiten möchte. Denken Sie an das Versenden einer E-Mail. In solchen Fällen sollten Sie nicht mit `ACTION_CHOOSER` arbeiten.

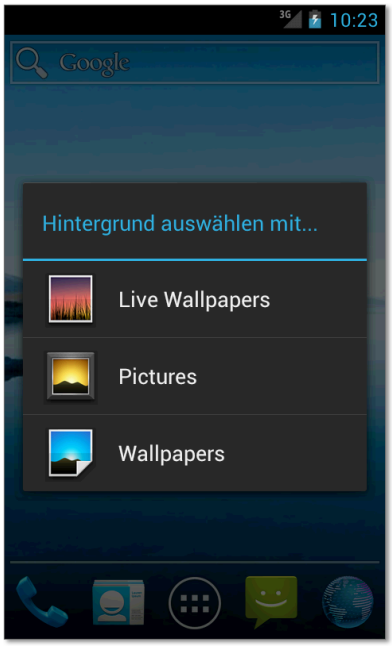


Abbildung 8.5 Action-Chooser

Anders verhält es sich, wenn der Anwender erwartet, sich entscheiden zu dürfen. Das ist der Fall, wenn er beispielsweise ein Foto aufgenommen hat und es an Freunde weitergeben möchte. Dies kann mittels E-Mail, MMS, Flickr oder beliebigen anderen Diensten geschehen. Hier wäre der Aufruf einer Standard-Activity ohne Rückfrage wahrscheinlich nicht im Sinne des Benutzers.

Sicher ist Ihnen aufgefallen, dass die App aus Abschnitt 8.2.1, »Die Wallpaper-API«, nicht in den beiden Auswahldialogen angezeigt wurde. Um dies zu erreichen, müssen Sie in deren Manifestdatei nur das `<activity />`-Element um einen Intent-Filter für `android.intent.action.SET_WALLPAPER` ergänzen. Das sieht folgendermaßen aus:

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

Listing 8.14 Setzen eines Intent-Filters in der Manifestdatei

Die Aktivität sollte mit `getIntent()` prüfen, ob ihr beim Start ein Intent übergeben wurde. Ist dies der Fall, kann dessen Action mit `getAction()` ermittelt werden. Wenn es sich um `ACTION_SET_WALLPAPER` handelt, können Sie programmatisch einen Hintergrund erstellen und zum Beispiel mit `setBitmap()` setzen.

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   Intent wp = getIntent();
   if ((wp == null) ||
      (!Intent.ACTION_SET_WALLPAPER.equals(
         wp.getAction()))) {
            Intent target = new
               Intent(Intent.ACTION_SET_WALLPAPER);
            startActivity(Intent.createChooser(target,
               "Hintergrund auswählen mit..."));
   } else {
      WallpaperManager m = WallpaperManager.getInstance(this);
      int w = m.getDesiredMinimumWidth();
      int h = m.getDesiredMinimumHeight();
      Bitmap bm = Bitmap.createBitmap(w, h, Config.RGB_565);
      Canvas c = new Canvas(bm);
      Paint paint = new Paint();
      paint.setTextAlign(Align.CENTER);
      paint.setColor(Color.WHITE);
      c.drawRect(0, 0, w - 1, h - 1, paint);
      paint.setColor(Color.BLUE);
      c.drawLine(0, 0, w - 1, h - 1, paint);
      c.drawLine(0, h - 1, w - 1, 0, paint);
      paint.setColor(Color.BLACK);
      c.drawText("Hallo Android!", w / 2, h / 2, paint);
      try {
         m.setBitmap(bm);
      } catch (IOException e) {
         Log.e("", "Fehler bei setBitmap()", e);
      }
   }
   finish();
}
```

Listing 8.15 Auszug aus WallpaperDemo.java

Sie finden das vollständige Projekt *WallpaperDemo* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

Abbildung 8.6 zeigt die Wallpaper-Auswahl, die auch den Hintergrund aus *WallpaperDemo.java* anbietet. In Abbildung 8.7 ist der Hintergrund selbst zu sehen.

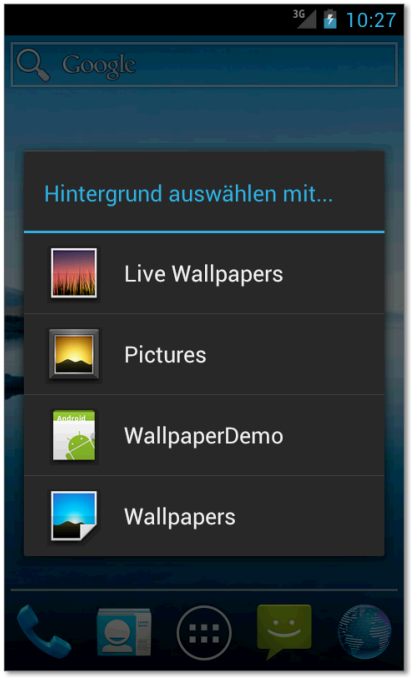


Abbildung 8.6 Wallpaper-Auswahl mit dem Wallpaper aus »WallpaperDemo«

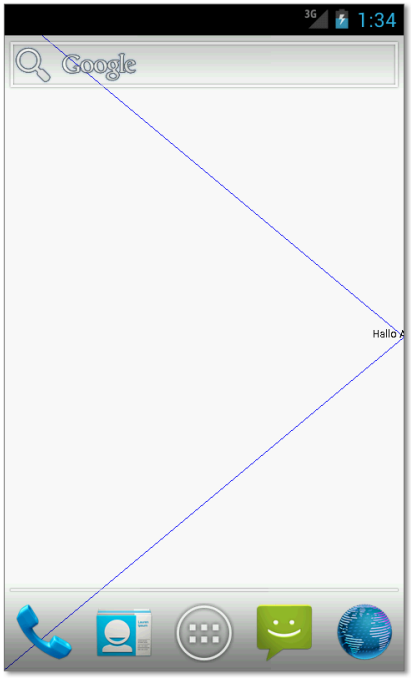


Abbildung 8.7 Der Hintergrund aus WallpaperDemo.java

Sie haben gesehen, dass es mit sehr wenig Aufwand möglich ist, Hintergründe programmatisch zu erstellen. Mit Live Wallpaper bietet Android aber noch faszinierendere Möglichkeiten. Diese möchte ich Ihnen im Folgenden vorstellen.

8.3 Live Wallpaper

Im Gegensatz zu klassischen Hintergründen sind Live Wallpapers animiert. Außerdem kann der Benutzer mit ihnen interagieren. Wie dies funktioniert, zeige ich Ihnen anhand der Beispiel-App *UhrzeitLiveWallpaper*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich, warten aber noch mit der Installation im Emulator bzw. auf dem Gerät.

Tipp

Wie Sie bereits wissen, sind Live Wallpapers seit Android 2.1 Bestandteil des Systems. Sie sollten beim Erstellen eigener animierter Hintergründe deshalb darauf achten, im Feld MIN SDK VERSION des Projektassistenten mindestens 7 einzutragen.

8.3.1 WallpaperService und Engine

Live Wallpapers werden als Services implementiert. Sie registrieren Ihre animierten Hintergründe also mit einem `<service />`-Element in der Manifestdatei Ihrer App. Damit Android weiß, dass es sich um ein Live Wallpaper handelt, setzen Sie einen Intent-Filter für die Aktion `android.service.wallpaper.WallpaperService`. Wichtig ist auch, die Berechtigung `android.permission.BIND_WALLPAPER` anzufordern.

```
<service
   android:label="@string/wallpaper_name"
   android:name=".UhrzeitLiveWallpaperService"
   android:permission="android.permission.BIND_WALLPAPER">
   <intent-filter>
      <action
        android:name="android.service.wallpaper.WallpaperService" />
   </intent-filter>
   <meta-data android:name="android.service.wallpaper"
     android:resource="@xml/wallpaper_uhrzeit" />
</service>
```

Listing 8.16 Definition eines Services in der Manifestdatei

Das Element `<meta-data />` verweist auf eine Konfigurationsdatei, die weitere Informationen zu einem Live Wallpaper enthalten kann. Sie wird unter *res/xml* abgelegt. Die Datei *wallpaper_uhrzeit.xml* des Projekts *UhrzeitLiveWallpaper* ist sehr einfach aufgebaut. Neben dem obligatorischen XML-Header enthält sie nur ein `<wallpaper />`-Element. Welche Kinder Sie hier einfügen können und wozu diese dienen, verrate ich Ihnen im folgenden Abschnitt.

```
<?xml version="1.0" encoding="utf-8"?>
<wallpaper
   xmlns:android="http://schemas.android.com/apk/res/android"
/>
```

Listing 8.17 Die Datei wallpaper_uhrzeit.xml

Wie Sie aus Kapitel 6, »Multitasking«, wissen, verweist das Attribut `android:name` des `<service />`-Elements auf eine Klasse, die den Service (also das Live Wallpaper) implementiert. Android beinhaltet `android.service.wallpaper.WallpaperService`. Von dieser abstrakten Basisklasse leiten Ihre animierten Hintergründe ab.

Auch `UhrzeitLiveWallpaperService` tut dies. Dieses in Abbildung 8.8 gezeigte Beispiel realisiert einen animierten Hintergrund, der die aktuelle Uhrzeit in zwei Farben und Schriftgrößen ausgibt. Die Position des Textes kann durch Antippen eines freien (schwarzen) Bereichs festgelegt werden. Die Farben für Stunden und Minuten wechseln alle 10 Sekunden.

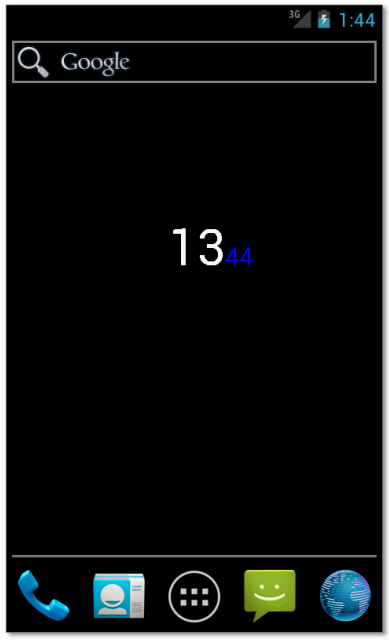


Abbildung 8.8 Live Wallpaper mit aktueller Uhrzeit

Die von `WallpaperService` abgeleitete Klasse selbst tut sehr wenig. Da die Methode `onCreateEngine()` abstrakt ist, müssen ihre Kinder diese implementieren. Sie liefert eine Instanz einer Klasse, die wiederum von `android.service.wallpaper.WallpaperService.Engine` erbt. Hier findet die eigentliche Arbeit statt.

Bevor ich ausführlicher darauf eingehe, möchte ich Ihnen nochmals vergegenwärtigen, dass Services an sich keine Nebenläufigkeit zur Verfügung stellen. Wie Sie aus Kapitel 6, »Multitasking«, wissen, laufen auch Services normalerweise im Haupt- oder Mainthread einer App. Deshalb gelten meine dortigen Hinweise zum Laufzeitverhalten auch für Live Wallpaper. Das bedeutet: Zeichenoperationen werden auf dem UI-Thread ausgeführt, aufwendige Berechnungen jedoch in individuell gestarteten Threads.

Um Farben und Uhrzeit regelmäßig aktualisieren zu können, nutzt `UhrzeitLiveWallpaperService` einen Handler. Dessen Methode `postDelayed()` ermöglicht die einmalige Ausführung einer `Runnable`-Instanz nach einer festgelegten Zeit in Millisekunden. Sicherlich fragen Sie sich, wie auf diese Weise eine Animation erfolgen kann. Nachdem sich unser Live Wallpaper das erste Mal gezeichnet hat, ruft es `postDelayed()` auf und initiiert damit einen erneuten Zeichenvorgang. Nach der angegebenen Zeit (10 Sekunden) beginnt der Vorgang von Neuem.

```
public class UhrzeitLiveWallpaperService extends WallpaperService {
   private final Handler handler = new Handler();

   @Override
   public Engine onCreateEngine() {
      return new UhrzeitLiveWallpaperEngine();
   }

   private class UhrzeitLiveWallpaperEngine extends Engine {
      private final Paint paint;
      private final Runnable runnable;
      private final Calendar cal;
      private int color;
      private float x, y;
      private boolean visible;

      UhrzeitLiveWallpaperEngine() {
         paint = new Paint();
         runnable = new Runnable() {
            public void run() {
               draw();
            }
         };
```

```
        cal = Calendar.getInstance();
    }

    @Override
    public void onCreate(SurfaceHolder surfaceHolder) {
        super.onCreate(surfaceHolder);
        setTouchEventsEnabled(true);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        handler.removeCallbacks(runnable);
    }

    @Override
    public void onVisibilityChanged(boolean v) {
        visible = v;
        if (v) {
            draw();
        } else {
            handler.removeCallbacks(runnable);
        }
    }

    @Override
    public void onSurfaceChanged(SurfaceHolder holder,
        int format,
        int width, int height) {
            super.onSurfaceChanged(holder,
            format, width, height);
        color = Color.WHITE;
        x = width / 2.0f;
        y = height / 2.0f;
        draw();
    }

    @Override
    public void onSurfaceDestroyed(SurfaceHolder holder) {
        super.onSurfaceDestroyed(holder);
        visible = false;
        handler.removeCallbacks(runnable);
    }
```

```
@Override
public void onTouchEvent(MotionEvent event) {
   super.onTouchEvent(event);
   if (event.getAction()
      == MotionEvent.ACTION_UP) {
         x = event.getX();
         y = event.getY();
         draw();
   }
}

void draw() {
   final SurfaceHolder holder = getSurfaceHolder();
   Canvas c = null;
   try {
      c = holder.lockCanvas();
      if (c != null) {
         int w = c.getWidth();
         int h = c.getHeight();
         paint.setColor(Color.BLACK);
         c.drawRect(0, 0, w - 1, h - 1, paint);
         cal.setTimeInMillis(System.currentTimeMillis());
         paint.setColor(color);
         paint.setTextSize(64);
         paint.setTextAlign(Align.RIGHT);
         c.drawText(String.format("%tH", cal),
            x, y, paint);
         color = getNextColor(color);
         paint.setColor(color);
         paint.setTextSize(32);
         paint.setTextAlign(Align.LEFT);
         c.drawText(String.format("%tM", cal),
            x, y, paint);
         color = getNextColor(color);
      }
   } finally {
      if (c != null)
         holder.unlockCanvasAndPost(c);
   }
   handler.removeCallbacks(runnable);
   if (visible) {
      handler.postDelayed(runnable, 10000);
   }
```

```
      }

      private int getNextColor(int color) {
         if (color == Color.WHITE) {
            return Color.BLUE;
         } else if (color == Color.BLUE) {
            return Color.RED;
         } else if (color == Color.RED) {
            return Color.GREEN;
         } else {
            return Color.WHITE;
         }
      }
   }
}
```

Listing 8.18 Auszug aus UhrzeitLiveWallpaperService.java

Aus Performancegründen dürfen sich animierte Hintergründe nur neu zeichnen, wenn sie sichtbar sind. Die Beispielimplementierung von `Engine` beinhaltet hierzu die `boolean`-Variable `visible`. Mit der Handler-Methode `removeCallbacks()` können ausstehende Aufrufe von `Runnable` beendet werden.

8.3.2 Live Wallpaper auswählen

Starten Sie nun das Projekt *UhrzeitLiveWallpaper* mit Run As • Android Application. Nach der Installation wird in der Sicht Console *Done!* ausgegeben. Anders als gewohnt, wird aber keine Activity geöffnet. Auch im Anwendungsstarter ist kein Symbol der App zu sehen. Um ein Live Wallpaper auszuwählen, tippen Sie auf eine freie Stelle des Home-Bildschirms und halten den Finger (bzw. die Maustaste) so lange gedrückt, bis sich ein Auswahlmenü öffnet. Wählen Sie Live Wallpapers (bzw. auf Geräten mit deutschsprachigen Meldungstexten Live-Hintergründe). Alle verfügbaren animierten Hintergründe werden in einer Liste zusammengefasst, die Sie in Abbildung 8.9 sehen.

Gerade neue Android-Nutzer sind mit der Auswahl von animierten Hintergründen nicht vertraut. Auch das fehlende Symbol im Programmstarter ist für viele ungewohnt. Zahlreiche exzellente Produkte haben in Google Play deshalb ungerechtfertigt schlechte Bewertungen erhalten. Um dies zu vermeiden, rate ich Ihnen, Ihrer Live-Wallpaper-App eine kleine Activity hinzuzufügen, die die Auswahl für animierte Hintergründe öffnet. Das folgende kleine Tutorial zeigt Ihnen, was Sie hierzu tun müssen.

Abbildung 8.9 Liste der installierten Live Wallpapers

Fügen Sie im Projekt *UhrzeitLiveWallpaper* zunächst der Datei *res/values/strings.xml* innerhalb des `<resources />`-Elements die Zeile

```
<string name="button">Live Wallpaper auswählen</string>
```

hinzu. Legen Sie nun unter *res/layout* die Datei *main.xml* an. Sie sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>
  <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button"
  />
</LinearLayout>
```

Listing 8.19 main.xml

Schließlich benötigen Sie noch die Klasse `com.thomaskuenneth.uhrzeitlivewallpaper.UhrzeitLiveWallpaper`. Deren Benutzeroberfläche besteht nur aus der Schaltfläche Live Wallpaper auswählen. Wird sie angeklickt, öffnet sich die Ihnen bereits bekannte Auswahl. Im Anschluss daran beendet sich die Activity.

```
package com.thomaskuenneth.uhrzeitlivewallpaper;

import android.app.Activity;
import android.app.WallpaperManager;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class UhrzeitLiveWallpaper extends Activity {
   private static final int RQ_PICKER = 1;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      Button b = (Button) findViewById(R.id.button);
      b.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent intent = new Intent();
            intent.setAction(
               WallpaperManager.ACTION_LIVE_WALLPAPER_CHOOSER);
               startActivityForResult(intent,
               RQ_PICKER);
         }
      });
   }

   @Override
   protected void onActivityResult(int requestCode,
      int resultCode, Intent data) {
         super.onActivityResult(requestCode, resultCode, data);
      if (requestCode == RQ_PICKER)
         finish();
   }
}
```

Listing 8.20 UhrzeitLiveWallpaper.java

Damit Android die Klasse `UhrzeitLiveWallpaper` als Hauptaktivität der App erkennt, müssen Sie noch ein paar Zeilen in die Manifestdatei übernehmen. Wie Sie bereits wissen, verweist das Attribut `android:name` des `<activity />`-Elements auf die Klasse.

```
<activity android:name=".UhrzeitLiveWallpaper"
          android:label="@string/app_name">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

Listing 8.21 UhrzeitLiveWallpaper als Hauptaktivität festlegen

Wenn Sie Ihren animierten Hintergrund über Google Play anbieten möchten, sollten Sie die Benutzeroberfläche dieser Hilfsactivity natürlich so ansprechend wie möglich gestalten. Erklären Sie dem Anwender, warum er die Schaltfläche anklicken soll und was dies bewirkt.

Tipp

Google Play nutzt das Manifest-Element `<uses-feature />`, um Apps nur für Geräte anzubieten, die eine bestimmte Funktion anbieten. Aus diesem Grund sollten Sie die Zeile `<uses-feature android:name="android.software.live_wallpaper" />` einfügen. Sie stellt sicher, dass Programme, die Live Wallpaper enthalten, nur auf hierfür geeigneten Tablets und Smartphones zu sehen sind.

8.3.3 Einstellungsseiten

Vielleicht fragen Sie sich, wie Sie Ihre animierten Hintergründe mit einer Einstellungsseite versehen können. Auf diese Weise ließe sich beispielsweise steuern, wie schnell sich die Farben in *UhrzeitLiveWallpaper* ändern. Mit einer von `android.preference.PreferenceActivity` abgeleiteten Klasse ist dies sehr einfach zu realisieren.

Wie Sie bereits wissen, kann Android Einstellungen mit `addPreferencesFromResource()` aus einer XML-Datei entfalten. Legen Sie als Erstes unter *res/values* die Datei *arrays.xml* an. Sie hat folgenden Inhalt:

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <string-array name="refresh_names">
        <item>"3 Sekunden"</item>
        <item>"10 Sekunden"</item>
        <item>"30 Sekunden"</item>
```

```
    </string-array>
    <string-array name="refresh_values">
        <item>3000</item>
        <item>10000</item>
        <item>30000</item>
    </string-array>
</resources>
```

Listing 8.22 arrays.xml

Die drei Elemente des Arrays `refresh_names` erscheinen später in einem Dialog, in dem der Benutzer die Anzeigedauer der zwei Farben auswählen kann. Das Feld `refresh_values` repräsentiert dieselben Werte, allerdings in maschinenlesbarer Form (Millisekunden). Erweitern Sie nun die Datei *strings.xml* um die folgenden Zeilen:

```
<string name="preferences_title">
   Einstellungen von UhrzeitLiveWallpaper
</string>
<string name="refresh_title">Anzeigedauer</string>
<string name="refresh_summary">
   Zeit in Sekunden, nach der ein Farbwechsel erfolgt
</string>
```

Jetzt können Sie die XML-Datei erstellen, die die Einstellungsseite beschreibt. Legen Sie sie unter *res/xml* ab.

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:title="@string/preferences_title"
   android:key="uhrzeitlivewallpaper">

    <ListPreference
            android:key="refresh"
            android:title="@string/refresh_title"
            android:summary="@string/refresh_summary"
            android:entries="@array/refresh_names"
            android:entryValues="@array/refresh_values"
    />

</PreferenceScreen>
```

Listing 8.23 preferences.xml

Die Datei *preferences.xml* wird von der Klasse `UhrzeitLiveWallpaperPreferences` entfaltet. Deren Quelltext besteht aus sehr wenigen Zeilen:

```
package com.thomaskuenneth.uhrzeitlivewallpaper;
import android.os.Bundle;
import android.preference.PreferenceActivity;
public class UhrzeitLiveWallpaperPreferences extends
   PreferenceActivity {
      @Override
      protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         addPreferencesFromResource(R.xml.preferences);
   }
}
```

Listing 8.24 UhrzeitLiveWallpaperPreferences.java

Sie haben es gleich geschafft. Wir müssen nur noch die Einstellungsseite als Aktivität in der Manifestdatei registrieren. Die hierfür nötigen Elemente kennen Sie bereits. Das Attribut `android:name` verweist, wie gehabt, auf die implementierende Klasse, `android:exported="true"` gibt an, dass die Activity auch von »fremden« Apps (oder dem System) gestartet werden darf. Mit `android:theme= "@android:style/Theme.WallpaperSettings"` teilen Sie der Plattform mit, dass Ihre Einstellungsseite wie alle Activities zum Konfigurieren von animierten Hintergründen aussehen soll.

```
<activity
   android:label="@string/preferences_title"
   android:name=".UhrzeitLiveWallpaperPreferences"
   android:theme="@android:style/Theme.WallpaperSettings"
   android:exported="true">
</activity>
```

Listing 8.25 Activity zur Konfiguration eines Live Wallpapers definieren

Erinnern Sie sich noch an die Datei *wallpaper_uhrzeit.xml*? Sie finden sie unter *res/xml*. Im vorigen Abschnitt hatte ich Sie vertröstet und versprochen, deren Kindelemente später vorzustellen. Fügen Sie ihr das Attribut `android:settingsActivity` hinzu, so dass die Datei folgendermaßen aussieht:

```
<?xml version="1.0" encoding="utf-8"?>
<wallpaper
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:settingsActivity="com.thomaskuenneth.
      uhrzeitlivewallpaper.UhrzeitLiveWallpaperPreferences"
/>
```

Listing 8.26 wallpaper_uhrzeit.xml

Auf diese Weise teilen Sie Android mit, welche Einstellungsseite zu einem Live Wallpaper gehört. Das Element `<wallpaper />` kennt auch die Attribute `android:thumbnail` und `android:description`. Diesen können Sie ein Vorschaubild (`@drawable/...`) sowie eine kurze Beschreibung des animierten Hintergrunds (`@string/...`) zuweisen.

Starten Sie das Projekt *UhrzeitLiveWallpaper*, und klicken Sie auf LIVE WALLPAPER AUSWÄHLEN. In der nun erscheinenden Liste müssen Sie UHRZEIT anklicken. Sie sehen daraufhin eine Vorschau des Live Wallpapers sowie zwei Schaltflächen zum Übernehmen dieses animierten Hintergrunds sowie zum Öffnen der Einstellungsseite. Diese ist in Abbildung 8.10 dargestellt.

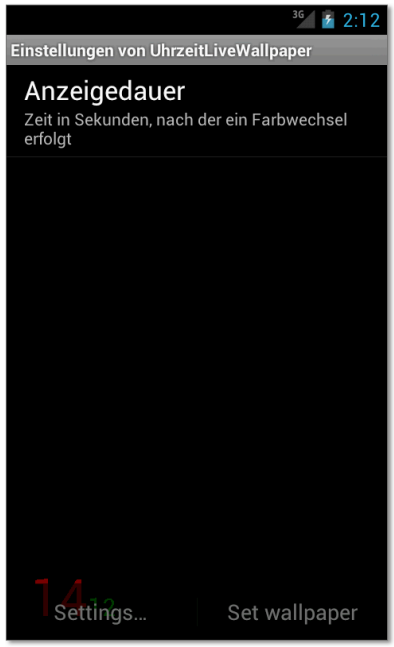


Abbildung 8.10 Einstellungsseite von »UhrzeitLiveWallpaper«

Leider hat die Auswahl einer Anzeigedauer noch keinen Effekt. Sie müssen in der Klasse `UhrzeitLiveWallpaperService` dafür sorgen, dass die durch `UhrzeitLiveWallpaperPreferences` gemachte Einstellung auch ausgelesen wird. In der Datei *preferences.xml* wird dem Attribut `android:key` des Elements `<ListPreference />` der Wert `refresh` zugewiesen. Auf dieses greifen Sie mit dem folgenden Quelltextfragment zu. Sie finden den Aufruf von `handler.postDelayed()` am Ende der Methode `draw()`:

```
if (visible) {
   SharedPreferences prefs =
      PreferenceManager
      .getDefaultSharedPreferences(
      UhrzeitLiveWallpaperService.this);
```

```
    String s = prefs.getString("refresh", "3000");
    handler.postDelayed(runnable, Integer.parseInt(s));
}
```

Sie finden die vollständige Version des Live Wallpapers *UhrzeitLiveWallpaper* als Projekt *UhrzeitLiveWallpaper_full* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

Kapitel 9
Sensoren und GPS

Android-Geräte enthalten zahlreiche Sensoren. Diese lassen sich mit geringem Aufwand in eigenen Apps nutzen. Wie das funktioniert, zeige ich Ihnen in diesem Kapitel.

Moderne Mobiltelefone schalten ihre Anzeige ab, sobald man sie in Richtung des Kopfes bewegt. Die Darstellung auf dem Bildschirm passt sich der Ausrichtung des Geräts an. Spiele reagieren auf Bewegungsänderungen. Karten-Apps erkennen automatisch den gegenwärtigen Standort. Und Restaurant- oder Kneipenführer beschreiben nicht nur den kürzesten Weg zur angesagten Döner-Bude, sondern präsentieren die Meinungen anderer Kunden und bieten Alternativen an. Dies und noch viel mehr ist möglich, weil die Android-Plattform eine beeindruckende Sensorenphalanx beinhaltet, die von allen Apps genutzt werden kann.

9.1 Sensoren

Android stellt seine Sensoren über eine Instanz der Klasse `SensorManager` zur Verfügung. Der Aufruf `getSystemService(SENSOR_SERVICE)` liefert eine entsprechende Referenz. Die Methode ist in allen von `android.content.Context` abgeleiteten Klassen vorhanden, beispielsweise in `android.app.Activity` und `android.app.Service`.

9.1.1 Die Klasse SensorManager

Nachdem Sie mit `getSystemService(SENSOR_SERVICE)` eine Referenz auf `SensorManager` ermittelt haben, können Sie auf verschiedene Weise prüfen, welche Sensoren in Ihrer App zur Verfügung stehen.

Vorhandene Sensoren ermitteln

Folgendes Quelltextfragment listet alle vorhandenen Sensoren auf:

```
List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ALL);
for (Sensor s : sensors) {
```

```
    Log.d(TAG, s.getName() + " (Hersteller: " + s.getVendor()
          + " , Version: " + s.getVersion() + ")");
}
```

Listing 9.1 Vorhandene Sensoren ermitteln

Anstelle von `TYPE_ALL` können Sie die übrigen mit `TYPE_` beginnenden Konstanten der Klasse `Sensor` nutzen, um nach einer bestimmten Art Ausschau zu halten. In so einem Fall ist es meist einfacher, stattdessen `getDefaultSensor()` aufzurufen.

Allerdings weist die Android-Dokumentation darauf hin, dass diese Methode unter Umständen einen Sensor liefert, der gefilterte oder gemittelte Werte produziert. Möchten Sie dies zum Beispiel aus Genauigkeitsgründen nicht, verwenden Sie `getSensorList()`. Neben ihren Namen und Herstellern liefern Sensoren Informationen zu ihrem Stromverbrauch (`getPower()`), ihrem Wertebereich (`getMaximumRange()`) und ihrer Genauigkeit (`getResolution()`).

Auf Sensorereignisse reagieren

Mit den beiden Methoden `registerListener()` und `unregisterListener()` der Klasse `SensorManager` können Sie sich über Sensor-Ereignisse informieren lassen sowie entsprechende Benachrichtigungen deaktivieren. `registerListener()` erwartet ein Objekt des Typs `SensorEventListener`.

Wie eine einfache Implementierung aussehen kann, zeigt die Klasse `SensorDemo1`. Sie finden das vollständige Projekt *SensorDemo1* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Die App nutzt den Helligkeitssensor eines Geräts und gibt je nach Helligkeit den gemessenen Wert oder den Text *sonnig* aus. Die Lebenszyklus-Methoden `onCreate()` und `onDestroy()` bzw. `onStart()` und `onPause()` einer Activity bieten sich an, um `SensorEventListener` zu registrieren bzw. zu entfernen.

```
public class SensorDemo1 extends Activity {
   private static final String TAG =
      SensorDemo1.class.getSimpleName();

   private TextView textview;
   private SensorManager manager;
   private Sensor sensor;
   private SensorEventListener listener;

   @Override
   public void onCreate(Bundle savedInstanceState) {

      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
```

```
    textview = (TextView) findViewById(R.id.textview);
    manager = (SensorManager)
       getSystemService(SENSOR_SERVICE);
    // Ist Helligkeitssensor vorhanden?
    sensor = manager.getDefaultSensor(Sensor.TYPE_LIGHT);
    if (sensor == null) {
       Log.d(TAG,
          "kein Helligkeitssensor vorhanden");
       // Activity beenden
       finish();
    }
    listener = new SensorEventListener() {
       @Override
       public void onAccuracyChanged(Sensor sensor,
          int accuracy) {
       }

       @Override
       public void onSensorChanged(SensorEvent event) {
          if (event.values.length > 0) {
             float light = event.values[0];
             String text =
                Float.toString(light);
             if (
                (SensorManager.LIGHT_SUNLIGHT <= light) &&
                (light <= SensorManager.LIGHT_SUNLIGHT_MAX)) {
                   text = "sonnig";
             }
             textview.setText(text);
          }
       }
    };
    // Listener registrieren
    manager.registerListener(listener, sensor,
       SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // Listener entfernen
    if (sensor != null) {
       manager.unregisterListener(listener);
```

```
        }
    }
}
```

Listing 9.2 Auszug aus SensorDemo1.java

Die Klasse `SensorManager` enthält zahlreiche Konstanten, die sich auf die vorhandenen Ereignistypen beziehen. Auf diese Weise können Sie, wie im Beispiel zu sehen ist, das Ergebnis der Helligkeitsmessung auswerten, ohne selbst in entsprechenden Tabellen nachschlagen zu müssen.

Welche Sensoren ein Android-Gerät oder der Emulator tatsächlich zur Verfügung stellt, können Sie erst zur Laufzeit Ihrer App prüfen. Selbstverständlich sollten Sie nicht einfach Ihre Activity beenden, wenn ein benötigter Sensor nicht zur Verfügung steht, sondern einen entsprechenden Hinweis ausgeben. Mithilfe des Elements `<uses-feature>` der Manifestdatei können Sie die Sichtbarkeit in *Google Play* auf geeignete Geräte einschränken. Hierzu ein Beispiel:

```
<uses-feature android:name="android.hardware.sensor.barometer"
   android:required="true" />
```

Apps, deren Manifest ein solches Element enthält, werden in *Google Play* nur auf Geräten angezeigt, die ein Barometer eingebaut haben. Beachten Sie hierbei aber, dass diese Filterung nicht die Installation verhindert, sofern die App auf anderem Wege auf das Gerät gelangt ist. Deshalb ist es wichtig, vor der Nutzung eines Sensors seine Verfügbarkeit wie weiter oben gezeigt zu prüfen.

9.1.2 Sensoren simulieren

Leider kann der Emulator nur sehr wenige Sensoren simulieren. Wenn Sie eine App entwickeln, die beispielsweise den Beschleunigungsmesser oder, wie *SensorDemo1*, den Helligkeitssensor benötigt, sind Sie deshalb auf ein echtes Gerät angewiesen. Glücklicherweise hat sich die Android-Community dieses Problems angenommen. Das Open Source-Projekt *Sensor Simulator* gaukelt Programmen, die im Emulator ausgeführt werden, das Vorhandensein zahlreicher Sensortypen vor.

Installation von Sensor Simulator

Sie können das Installationsarchiv von der Seite *Downloads* des Projekts *openintents* herunterladen.[1] Die zum Zeitpunkt der Drucklegung dieses Buches aktuelle Version *sensorsimulator-2.0-rc1.zip* finden Sie auch im Verzeichnis *Software* der Begleit-

1 *http://code.google.com/p/openintents/wiki/SensorSimulator*

DVD. Entpacken Sie das Archiv in einem beliebigen Ordner. Das Ergebnis ist in Abbildung 9.1 zu sehen.

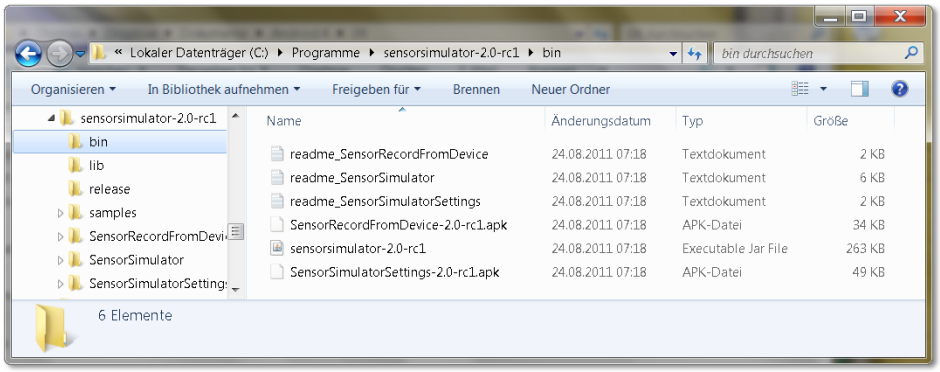


Abbildung 9.1 Inhalt des Sensor-Simulator-Archivs

Starten Sie nun den Sensor Simulator mit einem Doppelklick auf *bin/ sensorsimulator-2.0-rc1.jar*. Falls dies nicht funktioniert, können Sie in der Eingabeaufforderung bzw. Shell zunächst in den Unterordner *bin* des Verzeichnisses wechseln und `java -jar sensorsimulator-2.0-rc1.jar` eingeben.

Nun müssen Sie die App *SensorSimulatorSettings* in dem Emulator installieren, den Sie auch für das Testen Ihres Programms nutzen. Hierfür verwenden Sie die *Android Debug Bridge*. Wenn Sie die Verzeichnisse des Android SDKs dem Systempfad hinzugefügt haben, wie ich es in Kapitel 1, »Android – eine offene, mobile Plattform«, beschrieben habe, können Sie direkt in der Eingabeaufforderung bzw. einer Shell das Kommando `adb install SensorSimulatorSettings-2.0-rc1.apk` eingeben. Damit das klappt, müssen Sie sich im Ordner *bin* des Sensor-Simulator-Verzeichnisses befinden.

Achten Sie auch darauf, vorher den Emulator zu starten. Eine erfolgreiche Installation erkennen Sie an der Ausgabe `Success`. Starten Sie nun die App *Sensor Simulator*. Sie ist in Abbildung 9.2 zu sehen.

Auf der Registerkarte SETTINGS können Sie die IP-Adresse Ihres Entwicklungsrechners sowie das zu verwendende Socket eintippen. Erstere können Sie im linken Bereich des in Abbildung 9.3 gezeigten Sensor-Simulator-Hauptfensters ablesen.

Ist dies geschehen, verbinden Sie die PC-Anwendung mit dem Emulator, indem Sie auf die Registerkarte TESTING der App wechseln und CONNECT anklicken. Welche Sensoren aktiv sind, kontrollieren Sie auf der Registerkarte SENSORS des Sensor-Simulator-Hauptfensters. Die Sensorwerte stellen Sie auf der Registerkarte SENSOR PARAMETERS ein.

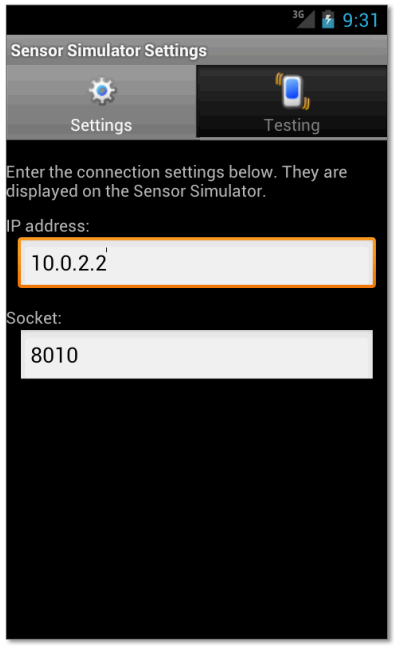


Abbildung 9.2 Die App »Sensor Simulator Settings«

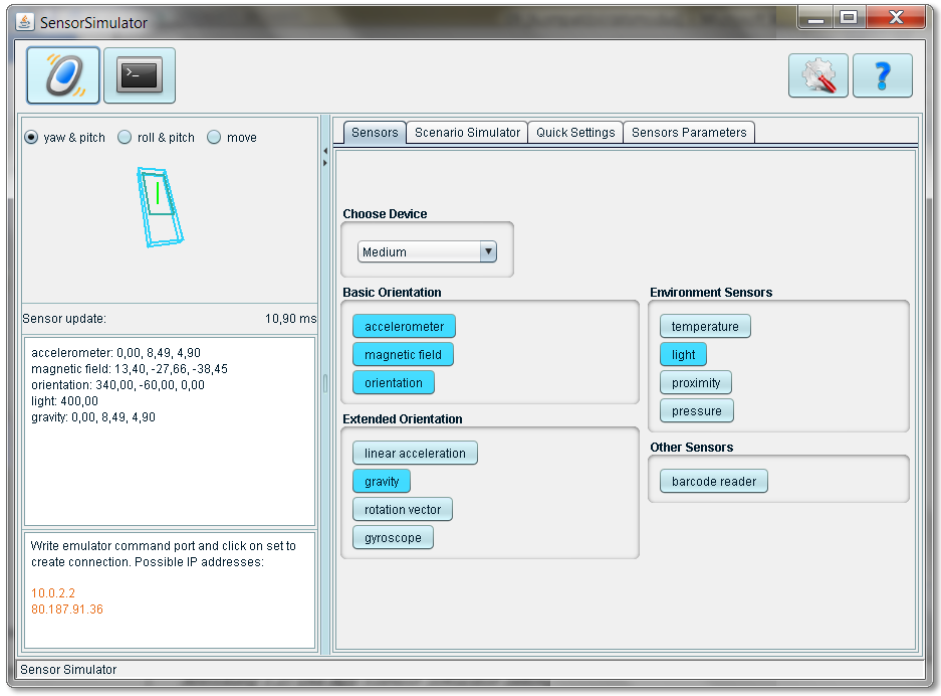


Abbildung 9.3 Hauptfenster des Sensor Simulators

Tipp

Um zu prüfen, ob die Verbindung zwischen PC und Emulator funktioniert, können Sie auf der Registerkarte Testing der App *Sensor Simulator Settings* ein Häkchen vor einem Sensor (beispielsweise der Temperatur) setzen. Werte, die Sie im Hauptfenster Ihres Entwicklungsrechners eingeben, müssen nach einer gewissen Verzögerung angezeigt werden. Für die im Folgenden beschriebene Integration in Ihre Apps ist dies aber nicht erforderlich.

Echte Sensordaten abfragen

Das Projekt *SensorDemo2* gibt die durch den Temperatursensor ermittelten Werte auf dem Bildschirm aus. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD. Die Klasse `SensorDemo2` registriert einen `SensorEventListener` und gibt in dessen Methode `onSensorChanged()` das Messergebnis aus.

```
package com.thomaskuenneth.sensordemo2;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class SensorDemo2 extends Activity {

   private static final String TAG =
      SensorDemo2.class.getSimpleName();

   private TextView textview;
   private SensorManager manager;
   private Sensor sensor;
   private SensorEventListener listener;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      textview = (TextView) findViewById(R.id.textview);
```

```
        manager = (SensorManager)
           getSystemService(SENSOR_SERVICE);

        sensor = manager.getDefaultSensor(Sensor.TYPE_TEMPERATURE);
        if (sensor == null) {
           Log.d(TAG, "kein Temperatursensor vorhanden");
           finish();
        }
        listener = new SensorEventListener() {
           @Override
           public void onAccuracyChanged(Sensor sensor, int accuracy) {
           }

           @Override
           public void onSensorChanged(SensorEvent event) {
              if (event.values.length > 0) {
                 float light = event.values[0];
                 String text = Float.toString(light);
                 textview.setText(text);
              }
           }
        };
        manager.registerListener(listener, sensor,
              SensorManager.SENSOR_DELAY_NORMAL);
     }

     @Override
     protected void onDestroy() {
        super.onDestroy();
        if (sensor != null) {
           manager.unregisterListener(listener);
        }
     }
}
```

Listing 9.3 SensorDemo2.java

Wenn Sie die App im Emulator starten, ist die Enttäuschung wahrscheinlich groß. Denn das Programm erkennt den simulierten Sensor nicht. Um dieses Problem zu beheben, sind jedoch nur wenige Schritte erforderlich.

Simulierte Sensordaten abfragen

Fügen Sie dem Projekt *SensorDemo2* das Archiv *sensorsimulator-lib-2.0-rc1.jar* hinzu. Sie finden es im Ordner *lib* des SensorSimulator-Verzeichnisses. Klicken Sie hierzu in der Sicht PACKAGE EXPLORER die Projektwurzel mit der rechten Maustaste an, und wählen Sie PROPERTIES. Navigieren Sie im sich öffnenden Eigenschaften-Dialog auf die Seite JAVA BUILD PATH. Sie ist in Abbildung 9.4 zu sehen. Mithilfe von ADD EXTERNAL JARS können Sie die benötigte Klassenbibliothek auswählen.

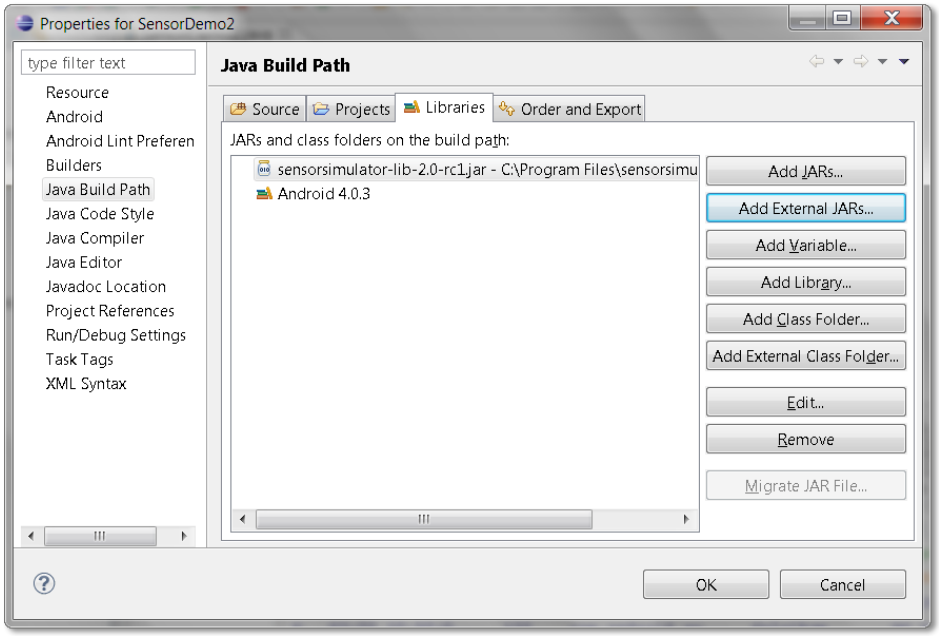


Abbildung 9.4 Dialog mit Projekt-Eigenschaften

Schließen Sie den Dialog mit OK, und nehmen Sie an der Klasse `SensorDemo2` die im Folgenden fett gesetzten Änderungen vor:

```
package com.thomaskuenneth.sensordemo2;
import org.openintents.sensorsimulator.hardware.Sensor;
import org.openintents.sensorsimulator.hardware.SensorEvent;
import org.openintents.sensorsimulator.hardware.SensorEventListener;
import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class SensorDemo2 extends Activity {
   private static final String TAG = SensorDemo2.class.getSimpleName();
```

```
private TextView textview;
private SensorManagerSimulator manager;
private Sensor sensor;
private SensorEventListener listener;

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   textview =
      (TextView) findViewById(R.id.textview);
   manager =
      SensorManagerSimulator.getSystemService(this, SENSOR_SERVICE);
   manager.connectSimulator();

   sensor = manager.getDefaultSensor(Sensor.TYPE_TEMPERATURE);
   if (sensor == null) {
      Log.d(TAG,
         "kein Temperatursensor vorhanden");
      finish();
   }
   listener = new SensorEventListener() {
      @Override
      public void onAccuracyChanged(Sensor sensor, int accuracy) {
      }

      @Override
      public void onSensorChanged(SensorEvent event) {
         if (event.values.length > 0) {
            float light = event.values[0];

            String text = Float.toString(light);
            textview.setText(text);
         }
      }
   };
   manager.registerListener(listener, sensor,
         SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onDestroy() {
```

```
      super.onDestroy();
      if (sensor != null) {
         manager.unregisterListener(listener);
      }
   }
}
```

Listing 9.4 SensorDemo2.java (geänderte Version)

Der Aufruf der Methode `connectSimulator()` ist neu. Bevor Sie die App starten, müssen Sie in der Manifestdatei noch die Berechtigung `android.permission.INTERNET` anfordern. Nun können Sie im Sensor-Simulator-Hauptfenster Temperaturwerte eintragen, die mit minimaler Verzögerung von *SensorDemo2* angezeigt werden.

9.2 GPS und ortsbezogene Dienste

Einer der Gründe für die große Beliebtheit von Smartphones und Tablets ist deren Fähigkeit, den aktuellen Standort ermitteln zu können. Wer in einer fremden Stadt schon einmal schnellstmöglich den Bahnhof erreichen oder einen Geldautomaten finden musste, möchte den Komfort, auf entsprechende Apps zurückgreifen zu können, sicherlich nicht mehr missen. Dabei sind die Einsatzgebiete dieser Technik ganz sicher noch lange nicht vollständig ausgelotet.

Haben Sie Lust bekommen, sich damit zu beschäftigen? In diesem Abschnitt zeige ich Ihnen, wie Sie die aktuelle Position ermitteln und auf einer Karte anzeigen können. Android bietet komfortable und einfach zu nutzende Programmierschnittstellen an. Lassen Sie Ihrer Kreativität freien Lauf.

9.2.1 Den aktuellen Standort ermitteln

Kopieren Sie das Projekt *LocationDemo1*, mit dem der Anwender seinen aktuellen Standort ermitteln kann, vom Verzeichnis *Quelltexte* der Begleit-DVD in Ihren Eclipse-Arbeitsbereich, und importieren Sie es anschließend. Die Klasse `LocationDemo1` ermittelt in der Methode `onCreate()` durch Aufruf von `getSystemService(LOCATION_SERVICE)` eine Instanz des Typs `android.location.LocationManager`. Dieses Objekt ermöglicht den Zugriff auf alle ortsbezogenen Funktionen des Systems. Sie können beispielsweise einen Listener registrieren, um unterrichtet zu werden, sobald sich der Standort des Geräts ändert.

LocationManager und LocationProvider

Positionsdaten werden durch *Location Provider* zur Verfügung gestellt. Um zu ermitteln, welche dieser Datenlieferanten vorhanden sind, können Sie die `LocationManager`-Methode `getAllProviders()` aufrufen. Sie liefert eine Liste mit den Namen der grundsätzlich verfügbaren Provider.

Mit `getProvider()` erhalten Sie eine Instanz des Typs `android.location.LocationProvider`. Der Rückgabewert von `isProviderEnabled()` gibt Auskunft darüber, ob der Anwender den korrespondierenden Provider auf der Einstellungsseite STANDORT UND SICHERHEIT ein- oder ausgeschaltet hat.

```
for (String name : providers) {
   LocationProvider lp = manager.getProvider(name);
   Log.d(TAG,
      lp.getName() +
      " --- isProviderEnabled(): " +
      manager.isProviderEnabled(name));
   Log.d(TAG, "requiresCell(): " + lp.requiresCell());
   Log.d(TAG, "requiresNetwork(): " + lp.requiresNetwork());
   Log.d(TAG, "requiresSatellite(): " + lp.requiresSatellite());
}
```

Listing 9.5 Informationen über vorhandene Location Provider ausgeben

Vielleicht fragen Sie sich, warum es mehrere Location Provider gibt. Sie können den Standort eines Geräts auf unterschiedliche Weise ermitteln. Die Nutzung des *Global Positioning Systems* (*GPS*) liefert recht genaue Positionen, funktioniert aber nur im Freien zuverlässig und benötigt vergleichsweise viel Strom.

Eine andere Möglichkeit besteht darin, Informationen von Sendemasten oder WIFI-Zugangspunkten auszuwerten. Das klappt natürlich nur, wenn das Tablet oder Smartphone in ein Netz eingebucht ist. Beide Varianten haben also spezifische Vor- und Nachteile. Android bietet deshalb die Möglichkeit, anhand von bestimmten Kriterien den am besten geeigneten Location Provider zu ermitteln.

Das folgende Beispiel liefert den Namen eines Providers, der die Position nur grob auflöst, dafür aber mit einem niedrigen Energieverbrauch auskommt:

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_COARSE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
String provider = manager.getBestProvider(criteria, true);
```

Listing 9.6 Provider mit geringem Verbrauch suchen

Mit dem Namen des am besten geeigneten Location Providers können Sie, wie gewohnt, die `LocationManager`-Methode `getProvider()` aufrufen.

Positionsänderungen abfragen

Es gibt zwei Möglichkeiten, den aktuellen Standort zu ermitteln. Die Methode `getLastKnownLocation()` des Location Managers liefert die letzte bekannte Position, die ein Location Provider ermittelt hat. Diese kann – muss aber nicht – dem aktuellen Aufenthaltsort entsprechen. Insofern bietet sich diese Methode vor allem an, um dem Anwender einen ersten Hinweis darauf zu geben, wo er sich befindet (oder zuletzt befunden hat). Beachten Sie aber, dass `getLastKnownLocation()` auch `null` liefern kann.

Die zweite Variante besteht darin, einen `LocationListener` zu registrieren. Dieser wird bei Positionsänderungen aufgerufen. Die hierfür zuständige Methode `requestLocationUpdates()` steht in zahlreichen Ausprägungen zur Verfügung. Beispielsweise können Sie steuern, wie viel Zeit mindestens zwischen zwei Aufrufen der Callback-Methode `onLocationChanged()` liegen sollte.

```
public class LocationDemo1 extends Activity {
   private static final String TAG = LocationDemo1.class.getSimpleName();
   private TextView textview;
   private LocationManager manager;
   private LocationListener listener;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      textview = (TextView) findViewById(R.id.textview);

      // LocationManager-Instanz ermitteln
      manager = (LocationManager)getSystemService(LOCATION_SERVICE);
      // Liste mit Namen aller Provider erfragen
      List<String> providers = manager.getAllProviders();

      // Infos zu Location Providern ausgeben
      for (String name : providers) {
         LocationProvider lp = manager.getProvider(name);
         Log.d(TAG,
               lp.getName() +
               " --- isProviderEnabled(): " +
               manager.isProviderEnabled(name));
```

```
      Log.d(TAG, "requiresCell(): " +
            lp.requiresCell());
      Log.d(TAG, "requiresNetwork(): " +
            lp.requiresNetwork());
      Log.d(TAG, "requiresSatellite(): " +
            lp.requiresSatellite());
   }

   // Provider mit grober Auflösung
   // und niedrigem Energieverbrauch
   Criteria criteria = new Criteria();
   criteria.setAccuracy(Criteria.ACCURACY_COARSE);
   criteria.setPowerRequirement(Criteria.POWER_LOW);
   // Namen ausgeben
   String name = manager.getBestProvider(criteria, true);
   Log.d(TAG, name);
   // LocationListener-Objekt erzeugen
   listener = new LocationListener() {
      @Override
      public void onStatusChanged(String provider,
            int status,
            Bundle extras) {
         Log.d(TAG, "onStatusChanged()");
      }

      @Override
      public void onProviderEnabled(String provider) {
         Log.d(TAG, "onProviderEnabled()");
      }

      @Override
      public void onProviderDisabled(String provider) {
         Log.d(TAG, "onProviderDisabled()");
      }

      @Override
      public void onLocationChanged(Location location) {
         Log.d(TAG, "onLocationChanged()");
         if (location != null) {
            String s = "Breite: " +
               location.getLatitude() +
               "\nLänge: " +
               location.getLongitude();
            textview.setText(s);
```

```
            }
         }
      };
   }

   @Override
   protected void onStart() {
      super.onStart();
      Log.d(TAG, "onStart()");
      manager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 3000, 0,
            listener);
   }

   @Override
   protected void onPause() {
      super.onPause();
      Log.d(TAG, "onPause()");
      manager.removeUpdates(listener);
   }
}
```

Listing 9.7 Auszug aus LocationDemo1.java

Die Klasse `LocationDemo1` registriert einen `LocationListener` für den GPS-basierten Location Provider. Wenn Sie das Projekt *LocationDemo1* im Emulator ausprobieren, können Sie Positionswechsel simulieren, indem Sie eine `telnet`-Verbindung aufbauen und das Kommando `geo fix <Länge> <Breite>` senden. Komfortabler ist die Eingabe allerdings über die *Emulator Control* der *DDMS*-Perspektive möglich.

Um auch die App *LocationDemo2*, die ich Ihnen im folgenden Abschnitt vorstellen werde, ausprobieren zu können, sollten Sie ein *Android Virtual Device* (AVD) mit API-Level 15 oder höher auf der Basis der Google-APIs erstellen. Sollte das virtuelle Gerät beim Versuch, Positionsdaten zu senden, abstürzen, rate ich Ihnen, ein AVD mit Google-API und API-Level 8 zu erzeugen. Meiner Erfahrung nach gibt es dann keine Probleme. Allerdings müssen Sie in diesem Fall den API-Level des Projekts von 15 auf 8 reduzieren.

Tipp

Je nachdem, welche bzw. welchen Location Provider Sie verwenden möchten, müssen Sie in der Manifestdatei Ihrer App die Berechtigungen `android.permission.ACCESS_COARSE_LOCATION` (Netzwerk) oder `android.permission.ACCESS_FINE_LOCATION` (GPS) anfordern.

Die Klasse android.location.Location

`getLastKnownLocation()` und `onLocationChanged()` liefern Instanzen der Klasse `android.location.Location`. Sie repräsentieren geografische Positionen (angegeben durch Länge und Breite) zu einem bestimmten Zeitpunkt. Informationen über Höhe, Geschwindigkeit und Richtung können zusätzlich vorhanden sein. Die Erde wird in 360 Längen- und 180 Breitengrade unterteilt. Da letztere vom Äquator aus gezählt werden, liegen die beiden Pole bei 90° Nord bzw. Süd. Der Nullmeridian teilt die Längengrade in westlicher und östlicher Richtung.

Innerhalb eines `Location`-Objekts werden Länge und Breite als `double` gespeichert. Die textuelle Darstellung hängt von der gewünschten Genauigkeit ab. Aus diesem Grund können Sie mit der Methode `convert()` eine Zeichenkette in eine Fließkommazahl umwandeln. Auch die andere Richtung ist möglich.

Hierzu ein Beispiel: Die ungefähre geografische Position von Nürnberg ist 49° 27' Nord und 11° 5' Ost. Grad und Minuten werden durch einen Doppelpunkt getrennt und als Zeichenkette an `convert()` übergeben:

```
Location locNuernberg = new Location(LocationManager.GPS_PROVIDER);
double latitude = Location.convert("49:27");
locNuernberg.setLatitude(latitude);
double longitude = Location.convert("11:5");
locNuernberg.setLongitude(longitude);
Log.d(TAG, "latitude: " + locNuernberg.getLatitude());
Log.d(TAG, "longitude: " + locNuernberg.getLongitude());
```

Listing 9.8 Umwandlung von String- in double-Werte

Soll aus einem `double`-Wert eine Zeichenkette bestehend aus Grad und Minuten erzeugt werden, übergeben Sie diesen an `convert()`. Der zweite Parameter ist `Location.FORMAT_MINUTES`.

9.2.2 Positionen in einer Karte anzeigen

In diesem Abschnitt zeige ich Ihnen, wie Sie den aktuellen Standort auf einer Karte visualisieren können. Die hierfür verwendete Klasse `com.google.android.maps.MapView` nimmt Ihnen die gesamte Kommunikation mit dem Dienst *Google Maps* ab. Allerdings ist sie nicht Teil der Standard-Android-Klassenbibliothek und damit nicht notwendigerweise auf jedem Android-Gerät vorhanden. Außen vor bleiben vor allem preiswertere Portable Media Player. Die meisten Smartphones und Tablets haben an dieser Stelle aber keine Probleme.

Der Maps-API-Schlüssel

Damit eine App Google Maps nutzen kann, muss sie einen Schlüssel übermitteln. Sie erhalten diesen kostenlos, indem Sie sich auf der Seite *http://code.google.com/intl/de-DE/android/add-ons/google-apis/maps-api-signup.html* registrieren. Im Rahmen dieser Registrierung müssen Sie den Nutzungsbedingungen des Dienstes Google Maps zustimmen.

Der Maps-API-Schlüssel ist an das Zertifikat gebunden, mit dem Sie Ihre Apps signieren. Während der Entwicklung verwenden Sie automatisch ein spezielles Debug-Zertifikat. Dieses befindet sich in der Datei *debug.keystore*. Wo diese abgelegt wird, hängt vom Betriebssystem Ihres Entwicklungsrechners ab. Unter Windows 7 ist dies der Ordner *.android* im Heimatverzeichnis des angemeldeten Benutzers.

Öffnen Sie die Eingabeaufforderung bzw. eine Shell, und wechseln Sie in das Verzeichnis, in dem *debug.keystore* liegt. Geben Sie nun die folgende Anweisung in einer Zeile ein. Sie erzeugt einen MD5-Fingerabdruck des Entwicklungszertifikats.

```
keytool -list -alias androiddebugkey -keystore debug.keystore
  -storepass android -keypass android
```

Damit das funktioniert, muss das Verzeichnis *bin* des Java Development Kits im Standardsuchpfad enthalten sein. Tragen Sie nun die 16 durch Doppelpunkt getrennten Hexadezimalwerte in das Eingabefeld der Registrierungsseite ein. Im Anschluss daran erhalten Sie von Google einen Maps-API-Schlüssel. Kopieren Sie diesen in eine beliebige Textdatei, oder speichern Sie die HTML-Seite auf Ihrem Rechner.

Tipp

Schlüssel, die aus dem Fingerabdruck eines Entwicklerzertifikats generiert wurden, funktionieren nur im Rahmen der Entwicklung. Wenn Sie eine App in Google Play anbieten möchten, die Google Maps nutzt, müssen Sie einen zusätzlichen API-Schlüssel aus dem Fingerabdruck Ihres Produktionszertifikats generieren.

MapViews anzeigen

Legen Sie in Eclipse ein neues Android-Projekt an, und nennen Sie es *LocationDemo2*. Tragen Sie `com.thomaskuenneth.locationdemo2` als Paketname ein, und lassen Sie die Activity `LocationDemo2` erzeugen. Wie Sie bereits wissen, gehört die Klasse `MapActivity` nicht zum Standardfunktionsumfang von Android, sondern ist Teil der *Google-APIs*. Aus diesem Grund setzen Sie bei BUILD TARGET ein Häkchen vor den Eintrag GOOGLE APIS mit dem API-Level 15. Wenn dieser Eintrag nicht angezeigt wird, installieren Sie das korrespondierende Paket bitte mit dem *Android SDK Manager*.

Nachdem Sie den Projektassistenten geschlossen haben, müssen Sie in der Manifestdatei noch eintragen, dass Ihre App die Google-Maps-Bibliothek verwendet und die Berechtigungen `android.permission.ACCESS_FINE_LOCATION` sowie `android.permission.INTERNET` benötigt.

Das Attribut `android:name` des Elements `<uses-library />` verweist auf das Paket `com.google.android.maps`. Damit können Sie Google Maps und die Klasse `MapActivity` in *LocationDemo2* nutzen.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.thomaskuenneth.locationdemo2"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        <uses-library android:name="com.google.android.maps" />

        <activity android:name=".LocationDemo2"
                  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 9.9 Manifestdatei des Projekts »LocationDemo2«

Der Inhaltsbereich einer `MapActivity` wird durch Aufruf der Methode `setContentView()` gesetzt. Die Layoutdatei *main.xml* von *LocationDemo2* ist sehr einfach gehalten. Das einzige Element `<MapView />` wird bildschirmfüllend angezeigt. Dessen Attribut `android:apiKey` muss den Google-Maps-API-Schlüssel enthalten, der Ihnen im Rahmen des Registrierungsprozesses übermittelt wurde.

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.maps.MapView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="...ihr Entwicklerschlüssel..."
/>
```

Listing 9.10 main.xml

Die Klasse `LocationDemo2` leitet von `com.google.android.maps.MapActivity` ab. In `onCreate()` wird das in der Datei *main.xml* abgelegte Layout entfaltet. Außerdem registriert die Activity einen `LocationListener`. Seine Methode `onLocationChanged()` packt den aktuellen Standort in eine Instanz des Typs `com.google.android.maps.GeoPoint` und übergibt ihn an die Methode `setCenter()` von einem `MapController`.

```
package com.thomaskuenneth.locationdemo2;

import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

public class LocationDemo2 extends MapActivity {
   private static final String TAG =
      LocationDemo2.class.getSimpleName();
   private LocationManager manager;
   private LocationListener listener;
   private MapController mapController;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
```

```
    // Location Manager ermitteln
    manager = (LocationManager)
       getSystemService(LOCATION_SERVICE);
    // LocationListener definieren
    listener = new LocationListener() {
       @Override
       public void onStatusChanged(String provider,
             int status,
             Bundle extras) {
          Log.d(TAG, "onStatusChanged()");
       }

       @Override
       public void onProviderEnabled(String provider) {
          Log.d(TAG, "onProviderEnabled()");
       }

       @Override
       public void onProviderDisabled(
             String provider) {
          Log.d(TAG, "onProviderDisabled()");
       }

       @Override
       public void onLocationChanged(
             Location location) {
          Log.d(TAG, "onLocationChanged()");
          // Koordinaten umwandeln
          int lat = (int) (location.getLatitude() * 1E6);
          int lng = (int) (location.getLongitude() * 1E6);
          GeoPoint point = new GeoPoint(lat, lng);
          mapController.setCenter(point);
       }
    };
    // Zoom aktivieren
    MapView mapView = (MapView) findViewById(R.id.mapview);
    mapView.setBuiltInZoomControls(true);
    // MapController ermitteln
    mapController = mapView.getController();
}
```

```
   @Override
   protected void onStart() {
      super.onStart();
      Log.d(TAG, "onStart()");
      manager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 3000, 0,
            listener);
   }

   @Override
   protected void onPause() {
      super.onPause();
      Log.d(TAG, "onPause()");
      manager.removeUpdates(listener);
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }
}
```

Listing 9.11 LocationDemo2.java

Die Anweisung `mapView.setBuiltInZoomControls(true)`; sorgt dafür, dass der Benutzer in die Karte hinein- bzw. herauszoomen kann. Die Klasse `MapView` (sie stellt die Karte letztlich dar) verwendet für Positionsangaben nicht die Ihnen aus dem vorherigen Abschnitt bereits bekannten Locations, sondern Instanzen von `GeoPoint`. Länge und Breite werden je als `int` gespeichert. Die Umrechnung erfolgt, indem Sie den `double`-Wert aus `Location` mit `1E6` multiplizieren.

Denken Sie daran, vor dem Start von *LocationDemo2* in der Datei *main.xml* Ihren Entwicklerschlüssel dem Attribut `android:apiKey` zuzuweisen. Nutzen Sie die Perspektive *DDMS*, um simulierte Positionsdaten an die in Abbildung 9.5 gezeigte App zu senden.

Weiterführende Informationen zur Nutzung von Google Maps und ortsbezogenen Diensten finden Sie im Dokument *Location and Maps*.[2]

2 *http://developer.android.com/guide/topics/location/index.html*

Abbildung 9.5 Die App »LocationDemo2«

TEIL IV

Dateien und Datenbanken

Kapitel 10
Das Android-Dateisystem

Das Lesen und Schreiben von Dateien gehört zu den Grundfunktionen vieler Apps. Wie dies unter Android funktioniert und worauf Sie beim Verwenden der Klassen und Methoden achten sollten, zeige ich Ihnen in diesem Kapitel.

Um Informationen längerfristig zu speichern, können Sie entweder Datenbanken oder klassische Dateien verwenden. Welche Variante Sie wählen, hängt von zahlreichen Faktoren ab. Termine und Kontakte sind sehr strukturierte Daten, das heißt, jeder »Datensatz« hat denselben Aufbau. Deshalb lassen sich solche Informationen sehr gut in relationalen Datenbanken ablegen. Musikstücke oder Videoclips hingegen haben eine weniger offensichtliche Struktur. Sie fühlen sich in herkömmlichen Dateien wohler.

Auch die Frage der Weitergabe spielt eine wichtige Rolle. Viele Android-Geräte haben einen Steckplatz für Speicherkarten. Informationen, die auf einem solchen Medium abgelegt wurden, lassen sich sehr leicht transportieren und in einem anderen Smartphone oder Tablet weiterverwenden. Haben Sie beispielsweise mit der eingebauten Kamera einen tollen Schnappschuss gemacht, können Sie durch einfaches Entnehmen der Speicherkarte vom Fotolabor einen Abzug anfertigen lassen.

10.1 Grundlegende Dateioperationen

Android erbt die Datei- und Verzeichnisoperationen von Java. Das Lesen und Schreiben von Dateien basiert also in weiten Teilen auf den Klassen und Interfaces das Pakets `java.io`. Wie Sie diese einsetzen, möchte ich Ihnen anhand der App *FileDemo1* demonstrieren. Sie sehen Sie später in Abbildung 10.1. Das Programm besteht aus einem Eingabefeld sowie den drei Schaltflächen LADEN, SPEICHERN und LEEREN. Mit ihnen wird der eingegebene Text gespeichert, geladen bzw. gelöscht. Sie finden das vollständige Projekt *FileDemo1* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

10.1.1 Dateien lesen und schreiben

Die für die beiden Schaltflächen Laden und Speichern registrierten `OnClickListener` rufen in ihren Implementierungen von `onClick()` die privaten Methoden `load()` bzw. `save()` auf. Letztere erhält als einzigen Parameter die zu speichernde Zeichenkette. Der Dateiname ist in der Konstanten `FILENAME` abgelegt. Sie wird folgendermaßen definiert:

```
private static final String TAG = FileDemo1.class.getSimpleName();
private static final String FILENAME = TAG + ".txt";
```

openFileOutput() und openFileInput()

Unter Java werden Daten in Ströme (Streams) geschrieben oder aus ihnen gelesen. Android stellt die beiden Methoden `openFileOutput()` und `openFileInput()` zur Verfügung, um Ströme für das Lesen oder Schreiben von Dateien zu öffnen. `openFileOutput()` benötigt zwei Parameter. Neben dem Namen der zu schreibenden Datei geben Sie an, ob nur die eigene App auf sie zugreifen darf oder ob auch Dritte lesen und schreiben dürfen. Existiert die Datei noch nicht, wird sie angelegt. Ist sie bereits vorhanden, geht der alte Inhalt verloren, sofern Sie nicht den Modus `MODE_APPEND` wählen. In diesem Fall »wächst« die Datei.

`openFileOutput()` liefert eine Instanz der Klasse `java.io.FileOutputStream`. Diese bietet einige `write()`-Methoden an, die allerdings auf Bytes operieren. Java setzt bei Zeichenketten auf Unicode, und einzelne Zeichen werden je in einem `char` abgelegt. Eine Umwandlung in Bytes ist mit der Methode `getBytes()` der Klasse `String` zwar prinzipiell möglich, allerdings kann es bei einer späteren Rückumwandlung Probleme geben, wenn der gespeicherte Block nicht in einem Stück eingelesen werden kann. Aus diesem Grund verlässt sich `FileDemo1` auf die Klasse `OutputStreamWriter`. Sie enthält eine Implementierung von `write()`, die Strings richtig verarbeitet:

```
private void save(String s) {
   FileOutputStream fos = null;
   OutputStreamWriter osw = null;
   try {
      fos = openFileOutput(FILENAME, MODE_PRIVATE);
      osw = new OutputStreamWriter(fos);
      osw.write(s);
   } catch (Throwable t) {
      // FileNotFoundException, IOException
      Log.e(TAG, "save()", t);
   } finally {
      if (osw != null) {
         try {
            osw.close();
```

```
            } catch (IOException e) {
               Log.e(TAG, "osw.close()", e);
            }
         }
         if (fos != null) {
            try {
               fos.close();
            } catch (IOException e) {
               Log.e(TAG, "fos.close()", e);
            }
         }
      }
}
```


Listing 10.1 Die Methode »save()« aus FileDemo1.java

Vielleicht fragen Sie sich, in welchem Verzeichnis die durch `openFileOutput()` erzeugte Datei abgelegt wird, denn ihr Name (*FileDemo1.txt)* enthält keine Pfadangaben. Die Methode `getFilesDir()` der Klasse `android.content.Context` liefert die gewünschte Information. Sie können in der Sicht File Explorer das Android-Dateisystem inspizieren.

```
File f = getFilesDir();
Log.d(TAG, "getFilesDir(): " + f.getAbsolutePath());
```


Tipp

Das Fangen der Ausnahme `IOException` beim Schließen eines Stroms wird leider viel zu oft unterlassen – was soll da schon passieren? Ich rate Ihnen, diese wenigen zusätzlichen Zeilen Quelltext keinesfalls wegzulassen. Sie machen Ihre App damit ein kleines Stück robuster.

Die private Methode `load()` lädt einen zuvor gespeicherten Text. Analog zu `openFileOutput()` liefert auch `openFileInput()` einen Strom, allerdings eine Instanz von `FileInputStream`. Da die `read()`-Methoden dieser Klasse keine `Strings` kennen, greife ich auf `java.io.InputStreamReader` und `java.io.BufferedReader` als Hüllen zurück:

```
private String load() {
   StringBuilder sb = new StringBuilder();
   FileInputStream fis = null;
   InputStreamReader isr = null;
   BufferedReader br = null;
   try {
      fis = openFileInput(FILENAME);
```

```
            isr = new InputStreamReader(fis);
            br = new BufferedReader(isr);
            String s;
            // Datei zeilenweise lesen
            while ((s = br.readLine()) != null) {
               // ggf. Zeilenumbruch hinzufügen
               if (sb.length() > 0) {
                  sb.append('\n');
               }
               sb.append(s);
            }
         } catch (Throwable t) {
            // FileNotFoundException, IOException
            Log.e(TAG, "load()", t);
         } finally {
            if (br != null) {
               try {
                  br.close();
               } catch (IOException e) {
                  Log.e(TAG, "br.close()", e);
               }
            }
            if (isr != null) {
               try {
                  isr.close();
               } catch (IOException e) {
                  Log.e(TAG, "isr.close()", e);
               }
            }
            if (fis != null) {
               try {
                  fis.close();
               } catch (IOException e) {
                  Log.e(TAG, "fis.close()", e);
               }
            }
         }
         return sb.toString();
      }
```

Listing 10.2 Die Methode »load()« aus FileDemo1.java

`load()` liest die Datei zeilenweise ein. Die einzelnen Teile werden durch Zeilenumbrüche miteinander verbunden und in einem `StringBuilder` gespeichert. Erst am Ende der Methode wird mit `toString()` ein String erzeugt. In Abbildung 10.1 sehen Sie die fertige App.

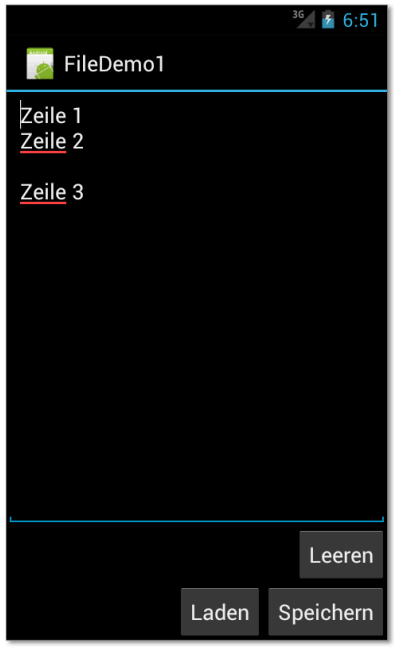


Abbildung 10.1 Die App »FileDemo1«

Dateiinformationen ermitteln

Da Android die Datei- und Verzeichnisfunktionen von Java erbt, ist es sehr einfach, beispielsweise die Länge einer Datei zu ermitteln oder sie zu löschen. Wie dies funktioniert, zeige ich Ihnen anhand der App *FileDemo2*. Wie üblich finden Sie das vollständige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD.

In der Methode `onCreate()` der Klasse `FileDemo2` lege ich zunächst zehn Dateien an. Ihr Name besteht aus dem Präfix *Datei_* und einer Zahl zwischen 1 und 10. Diese gibt auch die Länge in Bytes an. *Datei_7* ist also 7 Bytes groß. Das folgende Quelltextfragment zeigt, wie das Anlegen funktioniert:

```
for (int i = 1; i <= 10; i++) {
   FileOutputStream fos = null;
   String name = null;
   try {
      // ergibt Datei_1, Datei_2, ...
      name = "Datei_" + Integer.toString(i);
      fos = openFileOutput(name, MODE_PRIVATE);
```

```
            // ein Feld der Länge i mit dem Wert i füllen
            byte[] bytes = new byte[i];
            for (int j = 0; j < bytes.length; j++) {
               bytes[j] = (byte) i;
            }
            fos.write(bytes);
         } catch (Throwable t) {
            // FileNotFoundException, IOException
            Log.e(TAG, name, t);
         } finally {
            if (fos != null) {
               try {
                  fos.close();
               } catch (IOException e) {
               }
            }
         }
      }
```

Listing 10.3 Auszug aus FileDemo2.java

Die zu speichernden Daten werden in einem `byte`-Feld gesammelt. Alle Elemente haben den gleichen Inhalt, nämlich eine Zahl, die der Länge des Feldes und damit der Datei entspricht. Das Feld wird durch Aufruf der `FileOutputStream`-Methode `write()` geschrieben.

Wie Sie bereits wissen, können Sie mit `getFilesDir()` den Pfad des Verzeichnisses erfragen, in dem die mit `openFileOutput()` erzeugten Dateien abgelegt werden. Deren Namen erfahren Sie mit `fileList()`. Aus diesen beiden Informationen lässt sich ein `java.io.File`-Objekt bauen, um beispielsweise die Länge einer Datei zu ermitteln oder sie zu löschen:

```
// Dateien ermitteln
String[] files = fileList();
// Verzeichnis ermitteln
File dir = getFilesDir();
for (String name : files) {
   File f = new File(dir, name);
   // Länge in Bytes ermitteln
   Log.d(TAG, "Länge von " + name + " in Byte: " + f.length());

   // Datei löschen
```

```
    Log.d(TAG, "Löschen " + (f.delete() == false ? "nicht " : "")
          + "erfolgreich");
}
```

Listing 10.4 Auszug aus FileDemo2.java

Der Rückgabewert von `delete()` signalisiert, ob das Löschen erfolgreich war. Abbildung 10.2 zeigt die Ausgaben von *FileDemo2* in der Sicht LogCat.

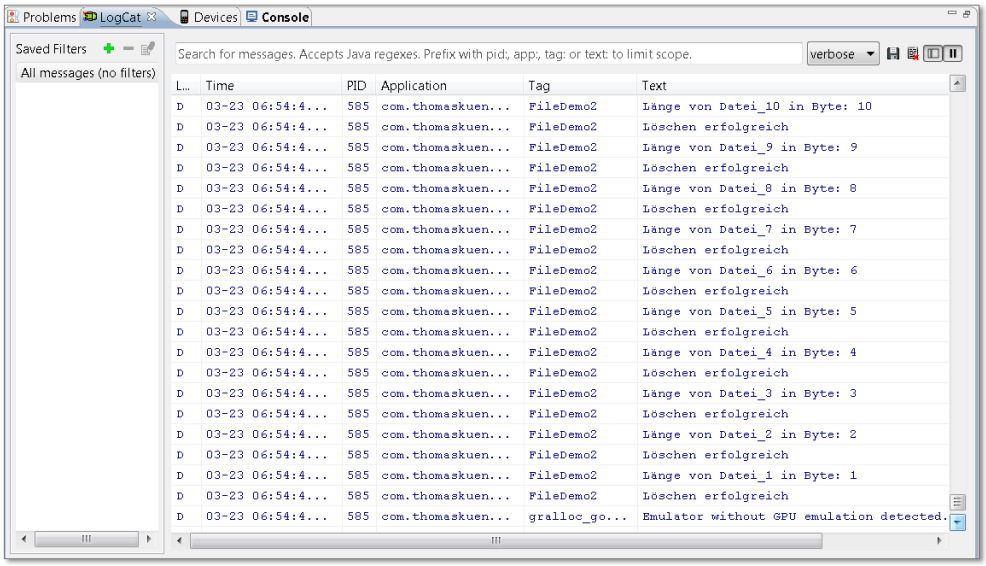


Abbildung 10.2 Die Ausgaben der App »FileDemo2« in der Sicht »LogCat«

Hinweis

Dateien und Verzeichnisse, die mit `openFileOutput()` bzw. `getDir()` erzeugt wurden, liegen innerhalb eines anwendungsspezifischen Basisverzeichnisses. Im Rahmen der Deinstallation werden diese gelöscht.

10.1.2 Mit Verzeichnissen arbeiten

Sofern eine App nur wenige Dateien erzeugt, ist das Ablegen in verschiedenen Verzeichnissen nicht nötig. Spielen Dateien jedoch eine zentrale Rolle, kann es sich lohnen, sie in unterschiedlichen Ordnern zu speichern.

Die Methode getDir()

`android.content.Context` bietet hierfür die Methode `getDir()`. Sie erhält als ersten Parameter den Namen eines Verzeichnisses. Sofern dieses noch nicht existiert, wird

es automatisch erzeugt. Der zweite Parameter steuert, wer auf das Verzeichnis zugreifen darf. Mögliche Werte sind `MODE_PRIVATE` oder `MODE_WORLD_READABLE` und `MODE_WORLD_WRITEABLE`.

Die App *FileDemo3* demonstriert, wie Sie `getDir()` verwenden. Sie finden das Projekt im Verzeichnis *Quelltexte* auf der Begleit-DVD. Das Programm erzeugt die beiden Dateien *A* und *B* im Standardverzeichnis der App. Wie Sie bereits wissen, können Sie dessen Pfad mit `getFilesDir()` ermitteln. Außerdem werden die beiden Verzeichnisse *audio* und *video* angelegt. Beide erhalten jeweils zwei Dateien, *C* und *D* bzw. *E* und *F*. `createFile()` ist eine private Methode von `FileDemo3`. Ihre Implementierung sehen Sie gleich in Listing 10.6.

```
// zwei leere Dateien erzeugen
createFile(getFilesDir(), "A");
createFile(getFilesDir(), "B");
// ein Verzeichnis erstellen
File dirAudio = getDir("audio", MODE_WORLD_READABLE);
// zwei leere Dateien erzeugen
createFile(dirAudio, "C");
createFile(dirAudio, "D");
// ein Verzeichnis erstellen
File dirVideo = getDir("video", MODE_WORLD_WRITEABLE);
// zwei leere Dateien erzeugen
createFile(dirVideo, "E");
createFile(dirVideo, "F");
```

Listing 10.5 Auszug aus FileDemo3.java

Zuvor möchte ich Ihnen noch die von Android aufgebaute Verzeichnisstruktur nach dem Start von *FileDemo3* zeigen. Sie können diese in der Sicht File Explorer überprüfen.

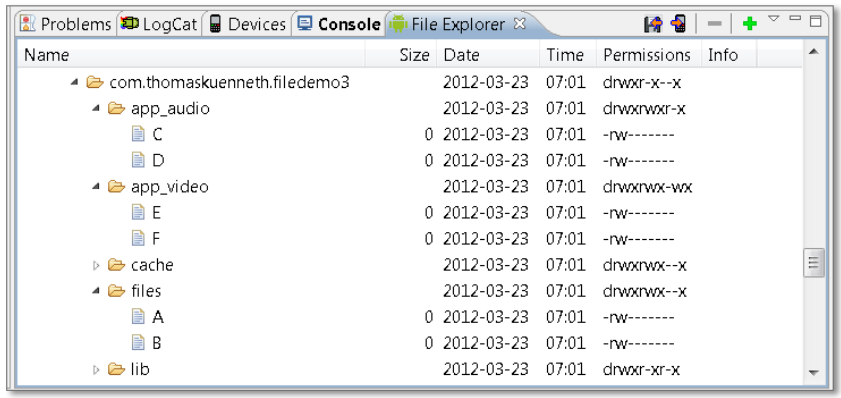


Abbildung 10.3 Von »FileDemo3« erzeugte Dateien und Verzeichnisse

Wie Sie in Abbildung 10.3 sehen, werden die Ordner *audio* und *video* nicht innerhalb desselben Verzeichnisses (*files*) erzeugt, in dem auch *A* und *B* liegen. Sie finden Sie stattdessen eine Ebene weiter oben, also in *com.thomaskuenneth.filedemo3*. Bitte beachten Sie in beiden Fällen das automatisch hinzugefügte Präfix *app_*. Hierbei handelt es sich allerdings um ein Android-internes Detail, das auf Ihre Programmierung keinen Einfluss hat.

Lassen Sie uns noch einen Blick auf die Implementierung von `createFile()` werfen. Sie nutzt nicht die aus dem vorherigen Abschnitt bekannte Methode `openFileOutput()`, weil ihr kein Verzeichnisname übergeben werden kann. Stattdessen wird direkt ein `FileOutputStream`-Objekt erzeugt. Hierzu verwenden Sie eine Instanz der Klasse `java.io.File`, der ein Verzeichnis sowie der Dateiname übergeben wurden:


```
private void createFile(File dir, String name) {
   FileOutputStream fos = null;
   try {
      File file = new File(dir, name);
      // Datei anlegen
      fos = new FileOutputStream(file);
   } catch (FileNotFoundException e) {
      Log.e(TAG, "openFileOutput()", e);
   } finally {
      // Datei schließen
      if (fos != null) {
         try {
            fos.close();
         } catch (IOException e) {
         }
      }
   }
}
```

Listing 10.6 Auszug aus FileDemo3.java

Temporäre Dateien

Es ist häufig nötig, temporäre Dateien zu erzeugen. Denken Sie an eine App, die Newsfeeds anzeigt. Um die Nachrichten darstellen zu können, muss die korrespondierende Datei zuerst von einem Server geladen werden. Sobald sie geparst wurde, wird sie jedoch nicht mehr benötigt. Solche kurzlebigen Dateien sollten nicht im Applikationsverzeichnis abgelegt werden.

Java bietet in der Klasse `java.io.File` die statische Methode `createTempFile()`. Sie erleichtert das Erzeugen von temporären Dateien und wird gerne in der folgenden Weise eingesetzt:

```
File file = null;
try {
   file = File.createTempFile("Datei_", ".txt");
} catch (IOException e) {
   Log.e(TAG, "", e);
} finally {
   if (file != null) {
      Log.d(TAG, file.getAbsolutePath());
   }
}
```

Listing 10.7 Erzeugen einer temporären Datei

Zwischen dem Präfix `Datei_` und dem Suffix `.txt` fügt das System einen mindestens fünf Zeichen langen automatisch erzeugten Teil ein. In welchem Verzeichnis die Datei angelegt wird, ergibt sich aus einem dritten Parameter, den die hier verwendete Zwei-Parameter-Variante auf `null` setzt. Dies führt dazu, dass die Java-System-Property `java.io.tmpdir` ausgewertet wird. Deren Wert können Sie mit der Anweisung

```
Log.d(TAG, System.getProperty("java.io.tmpdir"));
```

in der Sicht LogCat ausgeben. Unter Android ist dies */sdcard*. Der Zugriff auf externe Speichermedien ist allerdings mit der Berechtigung `android.permission.WRITE_EXTERNAL_STORAGE` abgesichert. Damit das Quelltextbeispiel keine `IOException` wirft, müssen Sie diese in der Manifestdatei Ihrer App anfordern:

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

Allerdings ist zu beachten, dass externe Speichermedien unter Umständen nicht verfügbar sind. Ausführliche Informationen hierzu finden Sie im folgenden Abschnitt. Android bietet deshalb in der Klasse `Context` die Methode `getCacheDir()`, die ebenfalls ein Verzeichnis für temporäre Dateien liefert – diesmal allerdings im internen Speicher des Geräts. Sie lässt sich sehr schön mit `createTempFile()` kombinieren:

```
Log.d(TAG, System.getProperty("java.io.tmpdir"));
Log.d(TAG, getCacheDir().getAbsolutePath());
File file = null;
try {
   file = File.createTempFile("Datei_", ".txt", getCacheDir());
} catch (IOException e) {
   Log.e(TAG, "", e);
} finally {
   if (file != null) {
```

```
        Log.d(TAG, file.getAbsolutePath());
    }
}
```

Listing 10.8 Erzeugen einer temporären Datei im internen Speicher

Tipp

Grundsätzlich kann das System Verzeichnisse für temporäre Dateien bei Bedarf selbstständig leeren. Allerdings sollten Sie als Entwickler sorgsam mit unter Umständen knappen Speicherplatz umgehen und nicht mehr benötigte Dateien möglichst sofort löschen.

10.2 Externe Speichermedien

Jedes Android-Gerät unterstützt zusätzlich zum internen Speicher ein »externes« Medium, auf dem alle Apps gleichberechtigt Daten ablegen können. Es kann sich hierbei um auswechselbare Medien (beispielsweise SD-Karten) oder um fest eingebauten Speicher handeln. Dateien, die dort abgelegt werden, können von jedermann gelesen und geschrieben werden.

10.2.1 Mit SD-Cards arbeiten

Im Unterschied zum internen Speicher müssen Wechselmedien und die auf ihnen abgelegten Informationen nicht permanent verfügbar sein. Aktiviert der Benutzer beispielsweise den USB-Massenspeichermodus seines Geräts, steht es als normales Laufwerk zur Verfügung. Für Android sind die Dateien in dieser Zeit allerdings nicht vorhanden. Aus diesem Grund sollten Sie vor Lese- oder Schreibzugriffen stets die Verfügbarkeit prüfen.

Hierfür gibt es die Klasse `android.os.Environment`. Sie stellt zahlreiche Auskunftsmethoden und Konstanten bereit, mit deren Hilfe Sie wichtige Pfade erfragen und den Status eines externen Mediums prüfen können. Beispielsweise liefert `isExternalStorageRemovable()` den Wert `true`, wenn der Anwender das Medium physikalisch entnehmen kann. `false` signalisiert, dass es fest eingebaut wurde.

Verfügbarkeit prüfen

Mit der Methode `getExternalStorageState()` ermitteln Sie seine aktuelle Verfügbarkeit. Die Klasse `Environment` definiert zahlreiche Konstanten, die Sie für Vergleiche mit `equals()` verwenden können. Das folgende Quelltextfragment ist Teil der Klasse

ExternalStorageDemo. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

```
final String state = Environment.getExternalStorageState();
final boolean canRead;
final boolean canWrite;
if (Environment.MEDIA_MOUNTED.equals(state)) {
   // lesen und schreiben möglich
   canRead = true;
   canWrite = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
   // lesen möglich, schreiben nicht möglich
   canRead = true;
   canWrite = false;
} else {
   // lesen und schreiben nicht möglich
   canRead = false;
   canWrite = false;
}
Log.d(TAG, "Lesen ist" +
      (canRead ? "" : " nicht") +
      " möglich");
Log.d(TAG, "Schreiben ist" +
      (canWrite ? "" : " nicht") +
      " möglich");
```

Listing 10.9 Auszug aus ExternalStorageDemo.java

Wenn Sie in Ihren Apps auf ein externes Medium zugreifen möchten, sollten Sie eine solche Abfrage implementieren. Nur wenn canRead bzw. canWrite den Wert true haben, können Sie Dateien lesen bzw. schreiben.

Verzeichnisse ermitteln

Damit Sie Dateien auf einem externen Medium ablegen können, müssen Sie zunächst Environment.getExternalStorageDirectory() aufrufen und damit den Zugriffspfad auf dessen Basisverzeichnis ermitteln. Diesem fügen Sie nach dem Muster */Android/data/<Paketname>/files* vier Unterverzeichnisse hinzu. Das folgende Quelltextfragment zeigt Ihnen, wie das funktioniert:

```
// Wurzelverzeichnis des externen Mediums
File dir1 = Environment.getExternalStorageDirectory();
Log.d(TAG, "getExternalStorageDirectory(): " +
      dir1.getAbsolutePath());
// App-spezifischen Pfad hinzufügen
```

```
File dirAppBase =
      new File(dir1.getAbsolutePath() + File.separator +
      "Android" + File.separator +
      "data" + File.separator +
      getClass().getPackage().getName() + File.separator +
      "files");
// ggf. Verzeichnisse anlegen
dirAppBase.mkdirs();
```

Listing 10.10 Auszug aus ExternalStorageDemo.java

Hinweis

`File.separator` liefert das Trennsymbol zwischen zwei Bestandteilen eines Pfads. Zwar ist es prinzipiell auch möglich, stattdessen den Slash (/) zu verwenden. Unter Java-Entwicklern ist der Zugriff auf die Konstante in `java.io.File` aber bewährte Praxis.

Ein nach diesem Muster angelegtes Verzeichnis wird von Android zu einer App gerechnet und ab Froyo (Version 2.2) bei ihrer Deinstallation mit seinem gesamten Inhalt gelöscht.

Wenn Sie das nicht möchten, können Sie mit der Methode `getExternalStoragePublicDirectory()` den Pfad auf einen gemeinsam genutzten Ordner ermitteln und Ihre Daten dort ablegen. Als Parameter übergeben Sie den gewünschten Typ, zum Beispiel `Environment.DIRECTORY_PICTURES`. Wie üblich, müssen Sie die Methode `mkdirs()` der zurückgelieferten `File`-Instanz aufrufen, um sicherzustellen, dass gegebenenfalls fehlende Verzeichnisse angelegt werden. Andernfalls riskieren Sie, dass zur Laufzeit Ihrer App Exceptions geworfen werden.

```
// Pfad auf Verzeichnis für Bilder
File dirPictures = Environment.getExternalStoragePublicDirectory
         (Environment.DIRECTORY_PICTURES);
// ggf. Verzeichnisse anlegen
dirPictures.mkdirs();
// Grafik erzeugen und speichern
FileOutputStream fos = null;
try {
   File file = new File(dirPictures, "grafik.png");
   fos = new FileOutputStream(file);
   saveBitmap(fos);
} catch (FileNotFoundException e) {

   Log.e(TAG, "new FileOutputStream()", e);
```

```
} finally {
   if (fos != null) {
      try {
         fos.close();
      } catch (IOException e) {
      }
   }
}
```

Listing 10.11 Auszug aus ExternalStorageDemo.java

Die Methode `saveBitmap()` erzeugt ein Objekt des Typs `Bitmap`, das aus zwei sich kreuzenden Linien und einem Text besteht. Anschließend wird die Grafik durch den Aufruf von `compress()` in einen `OutputStream` geschrieben:

```
private void saveBitmap(OutputStream out) {
   // Grafik erzeugen
   int w = 100;
   int h = 100;
   Bitmap bm = Bitmap.createBitmap(w, h, Config.RGB_565);
   Canvas c = new Canvas(bm);
   Paint paint = new Paint();
   paint.setTextAlign(Align.CENTER);
   paint.setColor(Color.WHITE);
   c.drawRect(0, 0, w - 1, h - 1, paint);
   paint.setColor(Color.BLUE);
   c.drawLine(0, 0, w - 1, h - 1, paint);
   c.drawLine(0, h - 1, w - 1, 0, paint);
   paint.setColor(Color.BLACK);
   c.drawText("Hallo Android!", w / 2, h / 2, paint);
   // und speichern
   bm.compress(CompressFormat.PNG, 100, out);
}
```

Listing 10.12 Auszug aus ExternalStorageDemo.java

Die vollständige Version von `ExternalStorageDemo` ist Teil des gleichnamigen Projekts. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD.

Hinweis

Apps, die auf das externe Medium schreibend zugreifen möchten, müssen in ihrer Manifestdatei die Berechtigung `android.permission.WRITE_EXTERNAL_STORAGE` anfordern.

10.2.2 Installationsort von Apps

Ab Android-Version 2.2 können Programme auf externen Speichermedien installiert werden. Besonders praktisch ist in diesem Zusammenhang, dass Apps sogar später noch zwischen dem internen Telefonspeicher und einer SD-Karte hin- und hergeschoben werden können. Die Einstellungsseite ist in Abbildung 10.4 zu sehen.

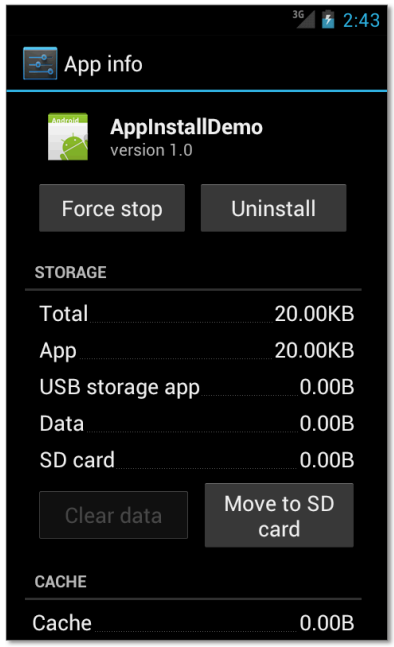


Abbildung 10.4 Eine App auf die SD-Karte verschieben

Vorbereitungen

Apps tragen in ihrer Manifestdatei den von ihnen bevorzugten Installationsort ein. Hierzu weisen Sie dem Attribut `android:installLocation` des Elements `<manifest />` einen der drei Werte `internalOnly`, `preferExternal` oder `auto` zu. `internalOnly` bedeutet, dass eine App nicht auf einem externen Medium installiert werden darf. Dies ist auch das Standardverhalten, wenn das Attribut `android:installLocation` nicht vorhanden ist.

Mit `preferExternal` signalisiert eine App, dass sie am liebsten auf einer SD-Karte installiert werden möchte. Ob das System diesem Wunsch nachkommt, hängt unter anderem davon ab, ob das Medium verfügbar und genug Speicherplatz vorhanden ist. `auto` überlässt Android die Entscheidung. In diesem Fall landet die App zunächst im internen Speicher, sofern er nicht zu voll ist. Dann nämlich wählt das System das externe Medium.

Hinweis

Bei der Installation auf einem externen Medium packt Android die *.apk*-Datei in einen verschlüsselten Container, der mit dem Gerät verknüpft ist. Wird eine SD-Karte mit Apps in ein anderes Gerät gesteckt, können die Programme auf diesem nicht benutzt werden.

Vielleicht fragen Sie sich, warum Sie die Installation auf einer SD-Karte unterbinden sollten. Wenn der Benutzer den USB-Massenspeichermodus aktiviert oder das Medium zum Beispiel über die Systemeinstellungen auswirft, werden alle aktuell laufenden Programme, die darauf installiert wurden, sofort beendet. Das Smartphone oder Tablet »vergisst« diese Apps, bis die Speicherkarte wieder eingelegt wird und für das System verfügbar ist.

Dies hat insbesondere Auswirkungen auf Widgets und Live Wallpapers. Sie werden vom Home-Bildschirm entfernt und müssen vom Benutzer manchmal manuell gesetzt bzw. hinzugefügt werden. Ein Neustart des Geräts stellt sie üblicherweise automatisch wieder her. Services werden gestoppt und nicht automatisch neu gestartet. Google empfiehlt deshalb, Apps, die solche Bausteine beinhalten, nicht für eine Installation auf externen Medien vorzusehen. Ausführliche Informationen hierzu finden Sie im Dokument *App Install Location*.[1]

Wiederverfügbarkeit abfragen

Vielerlei Gründe lassen es wünschenswert erscheinen, über die Wiederverfügbarkeit von auf der SD-Karte installierten Apps informiert zu werden. Denken Sie an alternative Home-Bildschirme. Diese könnten beispielsweise Live Wallpapers oder Appwidgets wiederherstellen. Android verschickt in einem solchen Fall ein Broadcast Intent mit der Aktion `android.intent.action.EXTERNAL_APPLICATIONS_AVAILABLE`. Um darauf zu reagieren, muss Ihre Klasse von `android.content.BroadcastReceiver` ableiten und die Methode `onReceive()` implementieren.

Eine Liste mit den Paketen, die wieder zur Verfügung stehen, kann mit `getStringArrayExtra(Intent.EXTRA_CHANGED_PACKAGE_LIST)` abgefragt werden. Die folgende Klasse ist Bestandteil des Projekts *AppInstallDemo*, das Sie im Verzeichnis *Quelltexte* der Begleit-DVD finden.

```
public class ExternalApplicationsAvailableReceiver
      extends BroadcastReceiver {
   private static final String TAG =
      ExternalApplicationsAvailableReceiver.class
         .getSimpleName();
```

1 *http://developer.android.com/guide/appendix/install-location.html*

```
   @Override
   public void onReceive(Context context, Intent intent) {
      if (intent != null) {
         if (Intent.ACTION_EXTERNAL_APPLICATIONS_AVAILABLE.equals(
            intent.getAction())) {
            String[] packages = intent.getStringArrayExtra(
                Intent.EXTRA_CHANGED_PACKAGE_LIST);
            for (String pkg : packages) {
               Log.d(TAG, pkg);
            }
         }
      }
   }
}
```

Listing 10.13 Auszug aus ExternalApplicationsAvailableReceiver.java

Damit `ExternalApplicationsAvailableReceiver` bei der Wiederverfügbarkeit von Apps aufgerufen wird, muss in der Manifestdatei von *AppInstallDemo* ein entsprechendes `<receiver />`-Element vorhanden sein:

```
<receiver android:name=".ExternalApplicationsAvailableReceiver"
      android:exported="true"
      android:enabled="true">
   <intent-filter>
      <action android:name="android.intent.action.
         EXTERNAL_APPLICATIONS_AVAILABLE" />
   </intent-filter>
</receiver>
```

Listing 10.14 Auszug aus der Manifestdatei der App »AppInstallDemo«

Um die App *AppInstallDemo* ausprobieren zu können, sollten Sie ein beliebiges anderes Programm mit dem Attribut `android:installLocation="preferExternal"` versehen und anschließend neu installieren. Nun können Sie über die Systemeinstellungen (Einstellungen • Speicher) die SD-Karte auswerfen. Nach einem erneuten Einhängen wird der Paketname der anderen App in der Sicht LogCat ausgegeben.

Hinweis

Sofern der Receiver zu einer auf einem externen Medium installierten App gehört, wird er bei Wiederverfügbarkeit *nicht* aufgerufen. Entgegen Googles Dokumentation ist es also bislang nicht möglich, auf diese Weise Services neu zu starten.

Kapitel 11
Datenbanken

Datenbanken eignen sich hervorragend, um viele gleichförmige Informationen abzuspeichern. Wie Ihre App Daten in Tabellen ablegen und mit SQL jederzeit darauf zugreifen kann, zeige ich Ihnen in diesem Kapitel.

Oft folgen die Daten, die mit einem Programm verarbeitet werden, einer wohl definierten Struktur. Kontakte beispielsweise enthalten fast immer einen Namen, eine Anschrift, eine oder mehrere Telefonnummern sowie ein Geburtsdatum. In einem Literaturverzeichnis erwarten Sie den Titel des Buches, den Namen des Verfassers und das Erscheinungsjahr. Und um Termine zu speichern, sollten dessen Beginn und Ende (jeweils mit Datum und Uhrzeit), der Ort und eine Beschreibung vorhanden sein. Meine drei Beispiele zeigen *Datensätze*. Ein Datensatz fasst *Datenfelder* zusammen. Name, Titel und Beschreibung sind solche Felder.

Wie oder wo Datensätze gespeichert werden, ist in erster Linie eine technische Frage. Indem Sie eine Karteikarte aus Papier beschriften und einsortieren, legen Sie einen Datensatz an. Programme können Sie in klassischen Dateien speichern. Wie das funktioniert, beschreibe ich in Kapitel 10, »Das Android-Dateisystem«. Welche Struktur hierbei entsteht, bestimmt die schreibende Anwendung. Möchten andere Programme darauf zugreifen, müssen sie das Format dieser Datei kennen. Eine interessante Alternative ist deshalb, die Speicherung einem Datenbanksystem zu überlassen. Android bringt eine solche Komponente mit. Deren Nutzung zeige ich Ihnen in diesem Kapitel.

11.1 Erste Schritte mit SQLite

Sehr häufig werden Datensätze als Tabellen dargestellt. Die Spaltenüberschriften repräsentieren Datenfelder. Die Zeilen der Tabelle enthalten die eigentlichen Nutzdaten. Dies ist die zentrale Idee von *relationalen Datenbanksystemen*. Sie folgen dem *relationalen Modell*, das E. F. Codd 1970 erstmals vorgeschlagen hat. Dessen Grundlage bildet die *mathematische Relation*. Relationen bestehen aus einer Menge von *Tupeln*. Ein solches Tupel wiederum entspricht einer Zeile einer Tabelle. Welche Ope-

rationen auf eine Relation angewendet werden können, wird durch die *relationale Algebra* bestimmt.

Glücklicherweise tritt dieser stark mathematische Aspekt in den Hintergrund, denn mit relationalen Datenbanken hat sich auch die Datenbanksprache *SQL* (*Structured Query Language*) etabliert. Mit ihr ist es unter anderem möglich, Daten abzufragen, zu verwalten und zu verändern.

11.1.1 Was ist SQLite?

Software zur Eingabe, Verwaltung und Bearbeitung von in Datenbanken abgelegten Informationen wird traditionell *Datenbankmanagementsystem* genannt. Oft werden die Programme, die die eigentliche Speicherung und Wiedergewinnung übernehmen, auf einem eigenen *Datenbankserver* installiert. Ein Client verbindet sich über ein Netzwerk mit ihm, übermittelt SQL-Anweisungen und empfängt Ergebnisse (zum Beispiel von Suchanfragen).

Damit das klappt, müssen beide Maschinen entsprechend konfiguriert, gegebenenfalls Benutzerkonten angelegt oder abgeglichen und Zugriffsrechte vergeben werden. Für den Einsatz im Unternehmen ist dieser Aufwand zweifellos gerechtfertigt. Möchten Sie in einer Anwendung »einfach nur« Daten speichern und abfragen, sind andere Qualitäten wichtig. Hierzu gehören unter anderem:

- ein geringer Speicherverbrauch
- eine möglichst einfache Konfiguration
- eine schlanke Programmierschnittstelle

SQLite[1] erfüllt diese Anforderungen. Es handelt sich hierbei um eine in sich geschlossene, serverlose und ohne Konfigurationsaufwand nutzbare transaktionale SQL-Datenbankmaschine. Google hat diese sehr kompakte (nur wenige 100 KB große) In-Process-Bibliothek in Android integriert und stellt sie App-Entwicklern über einige leicht einsetzbare Klassen zur Verfügung. Wie Sie in den folgenden Abschnitten sehen werden, gelingt der Einstieg in die faszinierende Welt der Datenbanken ohne große Mühe. Dennoch möchte ich Ihnen einen Blick in das Literaturverzeichnis in Anhang A ans Herz legen, wenn Sie tiefer in die Materie einsteigen möchten.

Wie Sie bereits wissen, werden Nutzdaten – zum Beispiel der Name einer Person und ihr Alter – als Zeilen einer benannten Tabelle gespeichert. Auch deren Spaltenüberschriften haben Namen. Das ist nötig, um eine ganz bestimmte Spalte ansprechen zu können. Welche Werte sie aufnehmen kann, wird bei der Definition der Tabelle festgelegt. Auch Datenbanken selbst erhalten – Sie ahnen es sicher – einen Namen.

1 *www.sqlite.org*

Nun fragen Sie sich bestimmt, wie Sie Datenbanken oder Tabellen anlegen und Inhalte einfügen, abfragen und verändern können. Vieles ist standardisiert über SQL möglich, anderes hängt vom verwendeten Datenbankmanagementsystem ab. Vor allem große Systeme kennen beispielsweise die Anweisung `CREATE DATABASE`, um eine neue Datenbank anzulegen. In SQLite entspricht eine Datenbank hingegen genau einer Datei. Deshalb müssen Sie eine Datei erzeugen, um mit einer neuen Datenbank zu arbeiten. Ein zusätzlicher Befehl ist nicht nötig.

11.1.2 Auf der Kommandozeile arbeiten

Bevor ich Ihnen zeige, wie Sie in Ihren Programmen auf SQLite zugreifen, möchte ich Ihnen ein paar grundlegende SQL-Kenntnisse vermitteln. (Falls Sie bereits mit SQL vertraut sind, können Sie diesen Abschnitt gefahrlos überspringen.) Da man sich Dinge, die man selbst ausprobiert hat, am leichtesten merken kann, werden Sie diese Befehle auf der Kommandozeile eingeben. Wie Sie bereits wissen, können Sie die *Android Debug Bridge* verwenden, um mit einem gestarteten Emulator zu kommunizieren.

Nachdem das AVD hochgefahren ist, öffnen Sie die *Eingabeaufforderung* oder eine *Shell*, und geben Sie `adb shell` ein. Damit das funktioniert, müssen Sie die Kommandozeilentools des Android SDK dem Standardsuchpfad hinzugefügt haben. Weitere Hinweise finden Sie in Abschnitt 1.3.1, »Android SDK«.

Datenbanken und Tabellen anlegen

Nun können Sie SQLite aufrufen. Geben Sie hierzu `sqlite3 /data/local/tmp/test.db` ein, und drücken Sie die ↵-Taste. */data/local/tmp/test.db* ist der vollständige Name der Datenbankdatei, mit der Sie arbeiten möchten. Da die Datei noch nicht existiert, legt das System sie für Sie an. Als Nächstes erzeugen Sie eine – zunächst leere – Tabelle. Hierfür ist `CREATE TABLE` zuständig. Geben Sie den folgenden Befehl in einer Zeile ein, und bestätigen Sie mit der ↵-Taste:

```
CREATE TABLE testtabelle (age INTEGER, name VARCHAR(32));
```

Achten Sie dabei auf den Strichpunkt am Ende. Ihre Eingabe besteht aus drei Bereichen:

1. der auszuführenden Operation (`CREATE TABLE`)
2. dem Namen der anzulegenden Tabelle
3. einer zwischen runde Klammern gesetzten Liste mit Spaltendefinitionen, deren Elemente durch ein Komma voneinander getrennt werden

Wie Sie bereits wissen, werden den Spalten einer Datenbanktabelle Namen und Datentypen zugewiesen. `testtabelle` besteht aus den beiden Spalten `age` und `name`.

Der Name einer Person kann in diesem Fall bis zu 32 Zeichen enthalten. Das Alter wird als ganze Zahl angegeben.

Datensätze ablegen

Mit `INSERT INTO` fügen Sie einer Tabelle Datensätze hinzu:

```
INSERT INTO testtabelle (age, name) VALUES(42, 'Thomas');
```

Ihre Eingabe besteht aus fünf Bereichen:

1. der auszuführenden Operation (`INSERT INTO`)
2. dem Namen der zu befüllenden Tabelle
3. einer zwischen runde Klammern gesetzten Liste mit Spaltennamen, deren Elemente durch ein Komma voneinander getrennt werden
4. dem Schlüsselwort `VALUES`
5. einer zwischen runde Klammern gesetzten Liste mit Werten, deren Elemente durch ein Komma voneinander getrennt werden

Die erste geklammerte Liste gibt an, welche Spalten einer Tabelle mit Werten gefüllt werden sollen. Die zweite Liste enthält die abzulegenden Daten. Der SQL-Befehl `INSERT INTO` ist sehr mächtig. Beispielsweise können Sie unter bestimmten Umständen Spalten auslassen. Solche fortgeschrittenen Themen werden in entsprechender Spezialliteratur behandelt. Die Lektüreliste in Anhang A enthält eine kleine Auswahl.

Daten abfragen

Mit `SELECT * FROM testtabelle;` können Sie sich die Tabelle ansehen. Die vollständige Sitzung mit allen bisher ausgeführten SQL-Anweisungen ist in Abbildung 11.1 zu sehen.

Anstelle des Asterisks (*) können Sie eine durch Kommas getrennte Liste von Spaltennamen übergeben. `SELECT age, name FROM testtabelle;` liefert hier also dasselbe Ergebnis. Fügen Sie der Tabelle noch ein paar Zeilen hinzu.

```
INSERT INTO testtabelle VALUES (44, 'Andy');
INSERT INTO testtabelle VALUES (73, 'Rüdel');
```

Hinweis

Derzeit bietet SQLite das von anderen Datenbankmanagementsystemen bekannte Einfügen mehrerer Datensätzen mit einer Anweisung `INSERT INTO` nicht an.

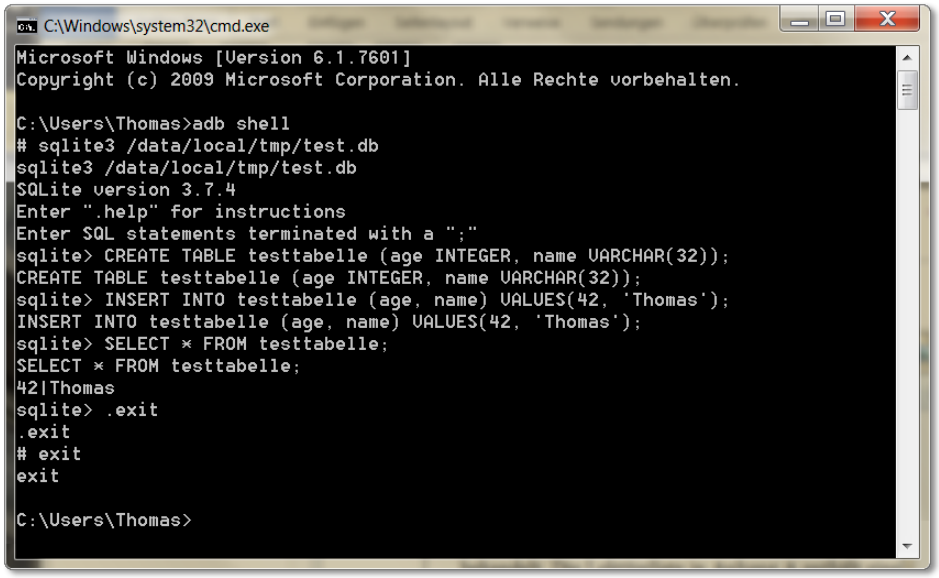


Abbildung 11.1 Eine sqlite3-Sitzung

Ist Ihnen aufgefallen, dass ich die zu befüllenden Spalten nicht ausdrücklich angegeben habe? Dies ist möglich, wenn die Liste nach `VALUES` vollständig ist und die Werte in derselben Reihenfolge wie bei der Definition der Tabellen übergeben werden.

Lassen Sie uns noch ein wenig mit SQL experimentieren. Um die Anzahl der Zeilen von `testtabelle` zu zählen, verwenden Sie `SELECT COUNT(*) FROM testtabelle;`. Das Durchschnittsalter der gespeicherten Personen erfragen Sie mit `SELECT AVG(age) FROM testtabelle;`. Geben Sie zum Schluss noch `SELECT age FROM testtabelle WHERE name IS 'Thomas';` ein. Diese Anweisung liefert das Alter einer Person mit dem Namen Thomas. Falls mehrere solcher Datensätze existieren (was natürlich problemlos möglich ist), werden entsprechend viele Zahlen ausgegeben.

Tipp

In Bezug auf Schlüsselwörter (`CREATE`, `SELECT`, ...) unterscheidet SQL nicht zwischen Groß- und Kleinschreibung. Allein schon aus Gründen der Lesbarkeit sollten Sie aber der Konvention folgen, diese in Großbuchstaben zu notieren, Bezeichner, zum Beispiel Tabellen- und Spaltennamen, hingegen in Kleinbuchstaben.

Sie haben jetzt erste Erfahrungen mit der Datenbanksprache SQL gesammelt und interaktiv mit SQLite gearbeitet. Wie Sie in Ihren Apps auf Datenbanken zugreifen, zeige ich Ihnen im folgenden Abschnitt. Vorher beenden Sie noch SQLite mit `.exit` (achten Sie auf den führenden Punkt). `exit` (diesmal ohne Punkt) trennt die `adb`-Verbindung.

11.1.3 SQLite in Apps nutzen

In diesem Abschnitt stelle ich Ihnen die Beispielapp *TKMoodley* vor, deren Benutzeroberfläche, wie Sie in Abbildung 11.2 sehen, aus vier Schaltflächen besteht – drei Smileys sowie VERLAUF.

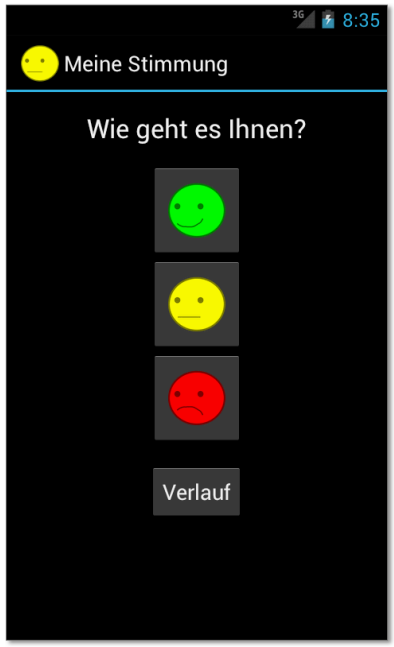


Abbildung 11.2 Die App »TKMoodley«

Die (nicht ganz ernst gemeinte) Idee ist, durch Anklicken eines der drei Gesichter Ihre aktuelle Stimmung zu dokumentieren. Das Programm legt den Zeitpunkt des Klicks sowie den Smiley-Typ in einer Datenbanktabelle ab.

Die Klasse TKMoodley

TKMoodley befindet sich in zwei Versionen auf der Begleit-DVD des Buches. Kopieren Sie fürs Erste nur *TKMoodley_v1* in Ihren Eclipse-Arbeitsbereich, und importieren Sie anschließend das Projekt. Es besteht aus den zwei Klassen `TKMoodley` und `TKMoodleyOpenHandler`, den drei Smileys in jeweils drei Größen sowie den Android-üblichen Layout- und Konfigurationsdateien. Die Activity *TKMoodley* entfaltet ihre Benutzeroberfläche aus main.xml. Wenn Sie neugierig sind, können Sie einen kurzen Blick auf die Datei werfen.

Die Methode `onCreate()` der Klasse `TKMoodley` registriert für die vier Schaltflächen jeweils einen `OnClickListener`. Alle Smileys rufen `imageButtonClicked()` auf. Diese private Methode erzeugt einen neuen Eintrag in einer Datenbank (wie das funktio-

niert, erkläre ich gleich) und gibt die Meldung Gespeichert in einem kleinen Infofähnchen (ein sogenanntes *Toast*) aus. Ein Klick auf Verlauf bewirkt im Moment noch nichts, da wir den entsprechenden Code erst in Abschnitt 11.2.1, »Klickverlauf mit SELECT ermitteln«, einfügen werden.

```
public class TKMoodley extends Activity {
   private TKMoodleyOpenHandler openHandler;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Benutzeroberfläche anzeigen
      setContentView(R.layout.main);

      // auf Anklicken des grünen Smileys reagieren
      final ImageButton buttonGut =
         (ImageButton) findViewById(R.id.gut);
      buttonGut.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            imageButtonClicked(TKMoodleyOpenHandler.MOOD_FINE);
         }
      });

      // auf Anklicken des gelben Smileys reagieren
      final ImageButton buttonOk =
         (ImageButton) findViewById(R.id.ok);
      buttonOk.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            imageButtonClicked(TKMoodleyOpenHandler.MOOD_OK);
         }
      });

      // auf Anklicken des roten Smileys reagieren
      final ImageButton buttonSchlecht =
         (ImageButton) findViewById(R.id.schlecht);
      buttonSchlecht.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            imageButtonClicked(TKMoodleyOpenHandler.MOOD_BAD);
         }
      });
```

```
        // auf Anklicken der Schaltfläche Auswertung reagieren
        final Button buttonAuswertung =
           (Button) findViewById(R.id.auswertung);
        buttonAuswertung.setOnClickListener(
           new OnClickListener() {
           @Override
           public void onClick(View v) {
              // hier muss noch was hin...
           }
        });
        openHandler = new TKMoodleyOpenHandler(this);
    }

    @Override
    protected void onPause() {
        super.onPause();
        openHandler.close();
    }

    private void imageButtonClicked(int mood) {
        openHandler.insert(mood, System.currentTimeMillis());
        // "Gespeichert"-Toast anzeigen
        Toast.makeText(this, R.string.saved,
           Toast.LENGTH_SHORT).show();
    }
}
```

Listing 11.1 Auszug aus TKMoodley.java

Die letzte Anweisung in `onCreate()` – `openHandler = new TKMoodleyOpenHandler(this)` ; – instanziiert ein Objekt des Typs `TKMoodleyOpenHandler` und weist es der Instanzvariablen `openHandler` zu. Auf diese wird in den Methoden `imageButtonClicked()` (Aufruf von `insert()`) und `onPause()` (Aufruf von `close()`) zugegriffen.

Die Klasse TKMoodleyOpenHandler

Damit ein Programm auf eine Datenbank zugreifen kann, muss diese unter anderem angelegt und geöffnet werden. Zu Android gehört die abstrakte Klasse `android.database.sqlite.SQLiteOpenHelper`. Sie vereinfacht die Kommunikation mit SQLite. Apps sollten deshalb eigene Datenbank-Hilfsklassen beinhalten, die von `SQLiteOpenHelper` ableiten und deren Methoden `onCreate()` und `onUpgrade()` implementieren.

Möchte eine App auf eine Datenbank zugreifen, prüft Android, ob diese schon existiert. Ist dies nicht der Fall, ruft das System `onCreate()` auf. In dieser Methode sollten

mit CREATE TABLE alle Tabellen angelegt und gegebenenfalls mit Grunddaten versorgt werden. Um das Anlegen der Datenbank selbst müssen Sie sich nicht kümmern.

onUpgrade() wird aufgerufen, wenn sich die Version einer Datenbank geändert hat. Diese Nummer übermitteln Sie bei der Instanziierung Ihrer SQLiteOpenHelper-Ableitung an das System. Benötigen Sie in einer Tabelle zusätzliche Spalten, oder müssen Sie die komplette Struktur Ihrer Datenbank ändern, erhöhen Sie einfach die Versionsnummer um 1.

Die Beispielimplementierung löscht mit dem SQL-Befehl DROP TABLE die einzige Tabelle der Datenbank und ruft anschließend onCreate() auf. Dies hat natürlich zur Folge, dass vorhandene Datensätze verloren gehen. Wenn Sie Ihre App in Google Play anbieten oder auf andere Weise veröffentlichen, sollten Sie diese in geeigneter Weise »parken«.

Tipp

Einfache Änderungen an der Tabellenstruktur sind mit ALTER TABLE möglich.

```
public class TKMoodleyOpenHandler extends SQLiteOpenHelper {
   private static final String TAG =
         TKMoodleyOpenHandler.class.getSimpleName();

   // Name und Version der Datenbank
   private static final String DATABASE_NAME = "tkmoodley.db";
   private static final int DATABASE_VERSION = 1;

   // Name und Attribute der Tabelle "mood"
   public static final String _ID = "_id";
   public static final String TABLE_NAME_MOOD = "mood";
   public static final String MOOD_TIME = "timeMillis";
   public static final String MOOD_MOOD = "mood";

   // Konstanten für die Stimmungen
   public static final int MOOD_FINE = 1;
   public static final int MOOD_OK = 2;
   public static final int MOOD_BAD = 3;

   // Tabelle mood anlegen
   private static final String TABLE_MOOD_CREATE
      = "CREATE TABLE "
         + TABLE_NAME_MOOD + " (" + _ID
         + " INTEGER PRIMARY KEY AUTOINCREMENT, "
         + MOOD_TIME + " INTEGER, "
```

```
      + MOOD_MOOD + " INTEGER);";
// Tabelle mood löschen
private static final String TABLE_MOOD_DROP =
      "DROP TABLE IF EXISTS "
      + TABLE_NAME_MOOD;

TKMoodleyOpenHandler(Context context) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL(TABLE_MOOD_CREATE);
}

@Override
public void onUpgrade(SQLiteDatabase db,
         int oldVersion, int newVersion) {
   Log.w(TAG, "Upgrade der Datenbank von Version "
         + oldVersion + " zu "
         + newVersion
         + "; alle Daten werden gelöscht");
   db.execSQL(TABLE_MOOD_DROP);
   onCreate(db);
}

public void insert(int mood, long timeMillis) {
   long rowId = -1;
   try {
      // Datenbank öffnen
      SQLiteDatabase db = getWritableDatabase();
      // die zu speichernden Werte
      ContentValues values = new ContentValues();
      values.put(MOOD_MOOD, mood);
      values.put(MOOD_TIME, timeMillis);
      // in die Tabelle mood einfügen
      rowId = db.insert(TABLE_NAME_MOOD, null, values);
   } catch (SQLiteException e) {
      Log.e(TAG, "insert()", e);
   } finally {
      Log.d(TAG, "insert(): rowId=" + rowId);
```

```
        }
    }
}
```

Listing 11.2 Auszug aus TKMoodleyOpenHandler.java

Die Methode `insert()` wird aus `TKMoodley` aufgerufen, wenn der Benutzer einen der drei Smileys angeklickt hat. Die Beispielimplementierung ermittelt zunächst mit `getWritableDatabase()` eine Instanz des Typs `SQLiteDatabase` und baut anschließend ein `ContentValues`-Objekt zusammen. Dieses erhält Informationen darüber, welchen Spalten welche Werte zugewiesen werden sollen.

Sehen Sie sich die Definition der Instanzvariable `TABLE_MOOD_CREATE` an. Sie enthält eine vollständige `CREATE TABLE`-Anweisung, mit der die Tabelle `mood` erstellt wird. Diese besteht aus den drei Spalten `_id`, `timeMillis` und `mood`. `_id` wird aufgrund des Schlüsselworts `AUTOINCREMENT` selbsttätig befüllt. Aus diesem Grund finden Sie in `insert()` nur zwei Aufrufe der Methode `put()`, nämlich einmal für `timeMillis` und einmal für `mood`.

Starten Sie die App *TKMoodley*, und klicken Sie einige Male auf die drei Smileys. Navigieren Sie anschließend in der in Abbildung 11.3 gezeigten Eclipse-Sicht FILE EXPLORER zum Verzeichnis */data/data/com.thomaskuenneth.tkmoodley_v1/databases*.

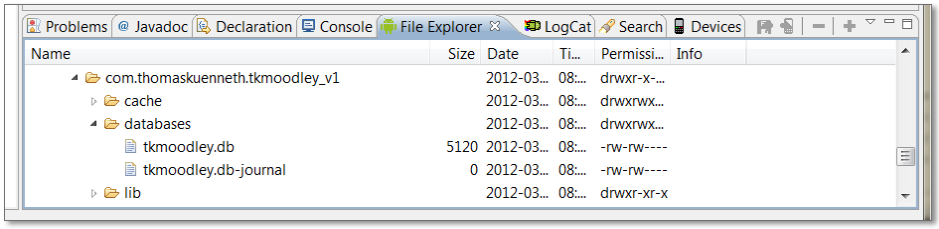


Abbildung 11.3 Die Sicht »File Explorer«

Dieses Verzeichnis enthält die Datei *tkmoodley.db*. Dies ist die Datenbank der App, einschließlich der Tabelle `mood` sowie der bislang erfolgten Klicks. Um sich deren Inhalt anzusehen, öffnen Sie die *Eingabeaufforderung* oder eine *Shell* und etablieren mit `adb shell` eine Verbindung zum Emulator.

Mit `sqlite3 /data/data/com.thomaskuenneth.tkmoodley_v1/databases/tkmoodley.db` starten Sie SQLite im interaktiven Modus. Ermitteln Sie mit `SELECT COUNT` die Anzahl Ihrer Klicks, oder lassen Sie sich wie in Abbildung 11.4 mit `SELECT *` die vollständige Tabelle `mood` anzeigen.

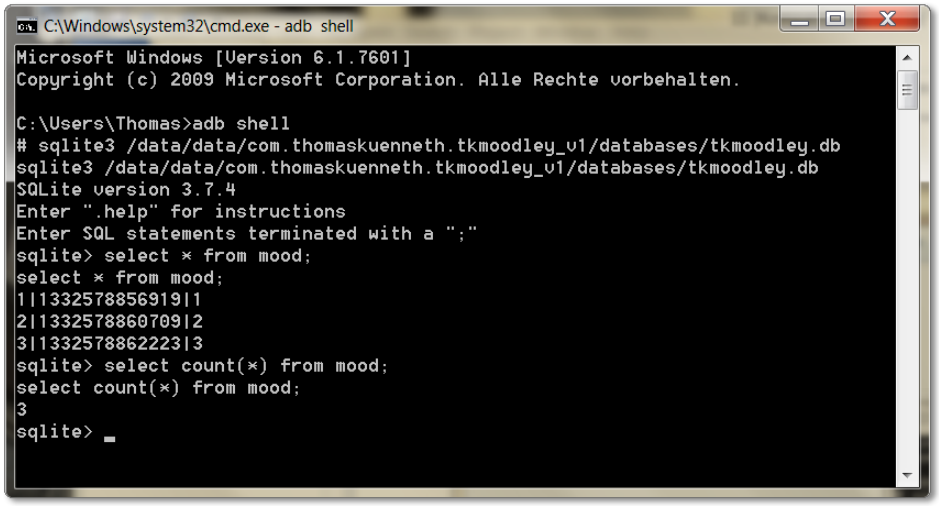


Abbildung 11.4 SQLite im interaktiven Modus

Tipp

Sie können mit der Sicht File Explorer Dateien zwischen dem Emulator und Ihrem Entwicklungsrechner austauschen. Das ist praktisch, um Sicherungskopien von Datenbanken zu erstellen und bei Bedarf wieder einzuspielen.

11.2 Fortgeschrittene Operationen

Das Erfassen von Stimmungen funktioniert schon tadellos. In diesem Abschnitt zeige ich Ihnen, wie Sie den Verlauf Ihrer Klicks anzeigen. Außerdem werden wir *TKMoodley* um die Möglichkeit erweitern, den Smiley-Typ nachträglich zu ändern und einen Eintrag komplett zu löschen.

Sie finden diese erweiterte Version unter *TKMoodley_full* im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Kopieren Sie das Projekt in Ihren Eclipse-Arbeitsbereich, und importieren Sie es anschließend. Ich habe den Paketnamen auf `com.thomaskuenneth.tkmoodley` geändert, so dass Sie die beiden Versionen parallel testen können. Wie Sie bereits wissen, werden die SQLite-Datenbanken im privaten Anwendungsverzeichnis abgelegt. Da der Paketname Bestandteil dessen Pfads ist, hat jede Variante der App ihre eigene Datenbank.

11.2.1 Klickverlauf mit SELECT ermitteln

TKMoodley zeigt den Klickverlauf in einer `ListActivity` an. Um diese starten zu können, tauscht `buttonAuswertung` die Implementierung der Methode `onClick()` des `OnClickListener` gegen folgendes Quelltextfragment aus:

```
public void onClick(View v) {
   Intent intent = new Intent(TKMoodley.this, History.class);
   startActivity(intent);
}
```

Listing 11.3 Auszug aus TKMoodley.java

Damit das funktioniert, müssen Sie die Activity wie üblich in der Manifestdatei bekannt machen. Das geschieht durch Hinzufügen des Elements `<activity android:name=".History" />` als Kind von `<application />`.

Mit CursorAdaptern arbeiten

List Activities bzw. List Views beziehen die anzuzeigenden Daten von Objekten, die das Interface `android.widget.ListAdapter` implementieren. Android enthält eine ganze Reihe von Adaptern, die Ihnen einen Großteil der sonst nötigen Implementierungsarbeit abnehmen. Beispielsweise stellt `android.widget.CursorAdapter` die Daten eines Cursors für List Views bereit. Warum dies in unserem Fall so praktisch ist, sehen Sie später.

Die Klasse `TKMoodleyAdapter` implementiert zwei Methoden: `newView()` wird vom System aufgerufen, wenn eine neue View benötigt wird. Dies ist zum Beispiel während der erstmaligen Befüllung einer Liste der Fall. Die Beispielimplementierung entfaltet die Layoutdatei *icon_text_text.xml*. Sie stellt zur Laufzeit ein Symbol sowie zwei Zeilen Text mit unterschiedlicher Größe dar (siehe Abbildung 11.5 im Hintergrund).

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="?android:attr/listPreferredItemHeight"
   android:padding="6dip">

   <ImageView android:id="@+id/icon"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginRight="6dip"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent" />
```

```
    <TextView android:id="@+id/text1"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:layout_marginTop="4dip"
        android:layout_toRightOf="@id/icon"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <TextView android:id="@+id/text2"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:layout_below="@id/text1"
        android:layout_alignLeft="@id/text1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Listing 11.4 icon_text_text.xml

Die zweite von `TKMoodleyAdapter` implementierte Methode, `bindView()`, überträgt konkrete Daten aus einem Cursor in eine bereits vorhandene View. Sie wird aufgerufen, wenn eine Liste befüllt wird oder wenn durch Scrollen verdeckte oder neue Bereiche sichtbar werden. Das Vorgehen ist stets gleich:

1. Auslesen eines Datenbankfeldes mit `cursor.getXYZ()`
2. Ermitteln der View mit `findViewById()`
3. Setzen des Wertes mit `view.setText()` oder Ähnlichem

Die Umsetzung dieser Schritte ist im folgenden Listing schön zu sehen:

```
public class TKMoodleyAdapter extends CursorAdapter {
    private final Date date;
    private static final DateFormat DF_DATE =
        SimpleDateFormat.getDateInstance(DateFormat.MEDIUM);
    private static final DateFormat DF_TIME =
        SimpleDateFormat.getTimeInstance(DateFormat.MEDIUM);
    private LayoutInflater inflator;
    private int ciMood, ciTimeMillis;

    public TKMoodleyAdapter(Context context, Cursor c) {
        super(context, c);
        date = new Date();
        inflator = LayoutInflater.from(context);
        ciMood =
            c.getColumnIndex(TKMoodleyOpenHandler.MOOD_MOOD);
```

```
      ciTimeMillis =
         c.getColumnIndex(TKMoodleyOpenHandler.MOOD_TIME);
   }

   @Override
   public void bindView(View view,
         Context context, Cursor cursor) {
      ImageView image =
         (ImageView) view.findViewById(R.id.icon);
      int mood = cursor.getInt(ciMood);
      if (mood ==
         TKMoodleyOpenHandler.MOOD_FINE) {
         image.setImageResource(
         R.drawable.smiley_gut);
      } else if (mood == TKMoodleyOpenHandler.MOOD_OK) {
         image.setImageResource(R.drawable.smiley_ok);
      } else {
         image.setImageResource(
            R.drawable.smiley_schlecht);
      }
      TextView textview1 =
         (TextView) view.findViewById(R.id.text1);
      TextView textview2 =
         (TextView) view.findViewById(R.id.text2);
      long timeMillis = cursor.getLong(ciTimeMillis);
      date.setTime(timeMillis);
      textview1.setText(DF_DATE.format(date));
      textview2.setText(DF_TIME.format(date));
   }

   @Override
   public View newView(Context context,
      Cursor cursor, ViewGroup parent) {
         return inflator.inflate(
            R.layout.icon_text_text, null);
   }
}
```

Listing 11.5 Auszug aus TKMoodleyAdapter.java

Wie Sie bereits wissen, speichert *TKMoodley* neben dem Smiley-Typ und dem Zeitpunkt der Erfassung eine eindeutige Kennung. Dieses `_id`-Feld wird vom System verwendet, um Tabellenzeilen auf Listenelementen abzubilden. Es muss vorhanden

sein, wenn Sie `CursorAdapter` nutzen möchten. In welchem Zusammenhang dies geschieht, zeige ich Ihnen im folgenden Abschnitt.

Die Klasse History

Die Klasse `History` leitet von `android.app.ListActivity` ab. Wenn der Benutzer einen Listeneintrag antippt und hält, erscheint ein Kontextmenü. Es ist in Abbildung 11.5 zu sehen.

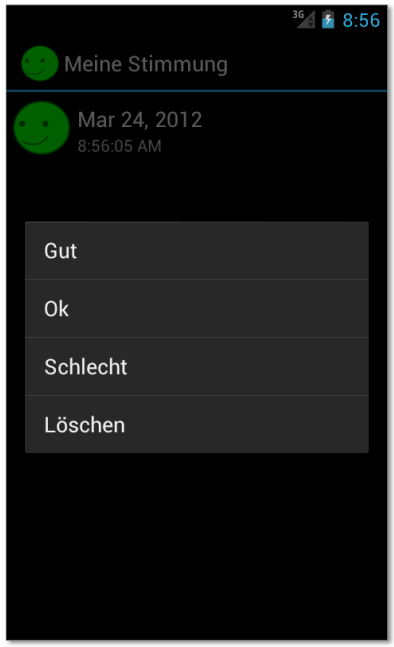


Abbildung 11.5 Das Kontextmenü der »Activity History«

Um dieses Kontextmenü in Ihren eigenen Apps zu realisieren, fügen Sie der Methode `onCreate()` die Anweisung `registerForContextMenu(getListView());` hinzu. Außerdem müssen Sie die Methode `onCreateContextMenu()` überschreiben. Implementierungen folgen sehr oft dem im folgenden Listing gezeigten Muster. Das Quelltextfragment entfaltet ein Menü aus der Datei *context_menu.xml*. Sie befindet sich in *res/menu*.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
      ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
```

```
    // Kontextmenü entfalten
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

Listing 11.6 Auszug aus History.java

Weitere Informationen zu Menüs finden Sie in Kapitel 5, »Benutzeroberflächen«.

Um auf das Auswählen eines Menübefehls zu reagieren, überschreiben Sie die Methode `onContextItemSelected()`. Üblicherweise werden in einem `switch-case`-Block alle Varianten sowie die gegebenenfalls auszuführenden Befehle aufgeführt. Wie das aussehen kann, zeige ich Ihnen im folgenden Abschnitt. Zunächst möchte ich aber zu der Frage zurückkommen, warum Cursor für die Klasse `History` eine so geeignete Datenquelle sind.

TKMoodley nutzt den Ihnen bereits bekannten `TKMoodleyOpenHandler` als Schnittstelle zu seiner Datenbank. Diese Klasse leitet von `android.database.sqlite.SQLiteOpenHelper` ab. Deren Methode `getWritableDatabase()` liefert ein Objekt des Typs `SQLiteDatabase`. Mit `query()` formulieren Sie Suchanfragen. Ergebnisse werden in Gestalt eines – Sie ahnen es sicher – Cursors übertragen. Das folgende Quelltextfragment liefert alle Zeilen der Tabelle `mood` in absteigender zeitlicher Reihenfolge. In der Listenansicht erscheinen neue Einträge also weiter oben.

```
public Cursor query() {
    // ggf. Datenbank öffnen
    SQLiteDatabase db = getWritableDatabase();
    return db.query(TABLE_NAME_MOOD,
        null, null, null,
        null, null,
        MOOD_TIME + " DESC");
}
```

Listing 11.7 Auszug aus TKMoodleyOpenHandler.java

Durch den Aufruf von `startManagingCursor(dbCursor);` in der Methode `onCreate()` von `History` wird dieser an den Lebenszyklus dieser Activity gekoppelt und beim Verlassen automatisch geschlossen. Da auch die Datenbank bei Nichtgebrauch geschlossen werden muss, überschreibt `History` auch die Methode `onDestroy()`.

```
public class History extends ListActivity {
    // Schnittstelle zur Datenbank
    private TKMoodleyOpenHandler dbHandler;
    // wird für die Listenansicht benötigt
    private Cursor dbCursor;
```

```
   // bildet den Cursor auf die ListView ab
   private TKMoodleyAdapter listAdapter;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Tippen und Halten öffnet Menü
      registerForContextMenu(getListView());
      dbHandler = new TKMoodleyOpenHandler(this);
      dbCursor = dbHandler.query();
      // Activity übernimmt Verwaltung des Cursors
      startManagingCursor(dbCursor);
      listAdapter = new TKMoodleyAdapter(this, dbCursor);
      setListAdapter(listAdapter);
   }

   @Override
   protected void onDestroy() {
      super.onDestroy();
      dbHandler.close();
   }
   ...
}
```

Listing 11.8 Auszug aus History.java

Im folgenden Abschnitt komme ich auf die Implementierung der Methode `onContextItemSelected()` zurück. Sie wird aufgerufen, wenn der Benutzer in der Activity *History* einen Befehl des Kontextmenüs auswählt. Dieses bietet an, einen neuen Smiley-Typ zu setzen oder den korrespondierenden Eintrag zu löschen.

11.2.2 Daten mit UPDATE ändern und mit DELETE löschen

Die Aktualisierung eines Eintrags findet in der Methode `update()` der Klasse `TKMoodleyOpenHandler` statt. Sie erhält als Parameter dessen eindeutige Kennung sowie den neuen Smiley-Typ. Die ID wird aus einer Instanz des Typs `AdapterContextMenuInfo` gelesen, die durch Aufruf der Methode `getMenuInfo()` ermittelt wurde. Die private Methode `updateList()` sorgt nach Änderungen an der Datenbank für eine Aktualisierung des Cursors sowie der `ListView`.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
   AdapterContextMenuInfo info = (AdapterContextMenuInfo)
```

```
      item.getMenuInfo();
   switch (item.getItemId()) {
   case R.id.menu_good:
      dbHandler.update(
         info.id, TKMoodleyOpenHandler.MOOD_FINE);
      updateList();
      return true;
   case R.id.menu_ok:
      dbHandler.update(
         info.id, TKMoodleyOpenHandler.MOOD_OK);
      updateList();
      return true;
   case R.id.menu_bad:
      dbHandler.update(
         info.id, TKMoodleyOpenHandler.MOOD_BAD);
      updateList();
      return true;
   case R.id.menu_delete:
      dbHandler.delete(info.id);
      updateList();
      return true;
   default:
      return super.onContextItemSelected(item);
   }
}

private void updateList() {
   // zunächst Cursor, dann Liste aktualisieren
   dbCursor.requery();
   listAdapter.notifyDataSetChanged();
}
```

Listing 11.9 Auszug aus History.java

Aktualisierung von Einträgen

Um einen oder mehrere Werte in einer Datenbank ändern zu können, müssen Sie ermitteln, welche dies sind. Die Änderung des Smiley-Typs bezieht sich immer auf genau eine Tabellenzeile, nämlich die zum Zeitpunkt der Kontextmenüauswahl aktive. Deren ID wird, wie Sie gerade gesehen haben, von Android zur Verfügung gestellt. Welche Werte geändert werden sollen, übermitteln Sie dem System mit einem Objekt des Typs `ContentValues`. Dessen Methode `put()` wird der Name der zu ändernden Spalte sowie der neue Wert übergeben.

Nun können Sie die SQLiteOpenHelper-Methode update() aufrufen. Sie erwartet den Namen einer Tabelle, eine Bedingung, die die zu ändernden Zeilen festlegt, sowie das Objekt mit den neuen Werten. Die Änderungsbedingung wird in zwei Teilen übergeben. _ID + " = ?" legt das Suchkriterium fest. Das Fragezeichen wird aus der Wertemenge String[] { Long.toString(id) } substituiert.

```
public void update(long id, int smiley) {
   // ggf. Datenbank öffnen
   SQLiteDatabase db = getWritableDatabase();
   ContentValues values = new ContentValues();
   values.put(MOOD_MOOD, smiley);
   int numUpdated = db.update(TABLE_NAME_MOOD,
         values, _ID + " = ?", new String[] { Long.toString(id) });
   Log.d(TAG, "update(): id=" + id + " -> " + numUpdated);
}
```

Listing 11.10 Aktualisieren eines Eintrags

Löschen eines Eintrags

Auch der zu löschende Eintrag wird in Gestalt seiner ID von Android übermittelt. Deshalb ist die Vorgehensweise beim Löschen analog zur Aktualisierung eines Datensatzes. Die SQLiteOpenHelper-Methode delete()erwartet den Namen einer Tabelle sowie eine Bedingung, die die zu löschenden Zeilen festlegt. Die Löschbedingung wird wiederum in zwei Teilen übergeben. _ID + " = ?" legt das Suchkriterium fest. Das Fragezeichen wird aus der Wertemenge String[] { Long.toString(id) } substituiert.

```
public void delete(long id) {
   // ggf. Datenbank öffnen
   SQLiteDatabase db = getWritableDatabase();
   int numDeleted = db.delete(TABLE_NAME_MOOD, _ID + " = ?",
         new String[] { Long.toString(id) });
   Log.d(TAG, "delete(): id=" + id + " -> " + numDeleted);
}
```

Listing 11.11 Auszug aus TKMoodleyOpenHandler.java

Kapitel 12
Content Provider

Mit Content Providern können Sie anderen Programmen die Daten Ihrer App zur Verfügung stellen. Warum das sehr praktisch ist und wie Sie diese Datenzugriffsschicht nutzen, zeige ich Ihnen in diesem Kapitel.

Android stellt eine Vielzahl von Funktionen zur Verfügung, um Dateien und Verzeichnisse zu lesen und zu schreiben. Üblicherweise werden diese im privaten Anwendungsverzeichnis abgelegt, sind also für andere Programme nicht erreichbar. Ist ein Austausch mit fremden Apps gewünscht, können Sie Dateien auf einer SD-Karte bzw. einem externen Speichermedium ablegen. Potenzielle Nutzer Ihrer Dateien müssen aber deren Aufbau kennen. Deshalb bietet sich ein Datenaustausch auf Dateiebene in der Regel nur für Standardformate an.

Ausführliche Informationen zum Umgang mit Dateien und Verzeichnissen finden Sie in Kapitel 10, »Das Android-Dateisystem«.

Strukturierte Daten sind besonders gut für eine Ablage in SQLite-Datenbanken geeignet. Auf diese greifen Sie mit der standardisierten Datenbanksprache SQL zu. Eigentlich prädestiniert sie dies für einen Zugriff durch Drittanwendungen. Allerdings sind auch Datenbanken für fremde Apps »unsichtbar«. Wie Sie aus Kapitel 11, »Datenbanken«, wissen, werden sie in einer Datei im privaten Anwendungsverzeichnis gespeichert. Vielleicht fragen Sie sich nun, wie Sie unter Android solche tabellenartigen Daten über Anwendungsgrenzen hinweg austauschen können.

In Kapitel 7, »Rund ums Telefonieren«, habe ich Ihnen die Nutzung der systemweiten Anrufhistorie (Call Log) gezeigt. Mit der Methode `query()` einer `ContentResolver`-Instanz wurden alle entgangenen Anrufe ermittelt und in einer Liste abgelegt. Hierzu haben wir in einer Schleife einen `Cursor` zeilenweise weiter bewegt und auf einzelne Ergebnisspalten zugegriffen. Auch das Verändern von Werten mit der Methode `update()` wurde demonstriert.

Hinweis

`Cursor`, `query()` und `update()` klingen zwar nach SQL, gehören in diesem Fall aber zu einer weiteren Datenzugriffsschicht von Android. Sie ist Gegenstand dieses Kapitels. Solche *Content Provider* stellen Informationen als Datensätze zur Verfügung. Alle interessierten Apps können mit einem *Content Resolver* auf diese zugreifen.

12.1 Vorhandene Content Provider nutzen

Neben dem Call Log enthält Android einige weitere interessante Content Provider. Wie Sie gleich sehen werden, erfolgen Zugriffe stets nach demselben Muster. Zuerst ermitteln Sie die Referenz auf ein Objekt des Typs `android.content.ContentResolver`.

Üblicherweise geschieht dies durch Aufruf von `getContentResolver()`. Diese Methode ist in allen von `android.content.Context` abgeleiteten Klassen vorhanden. Anschließend nutzen Sie `query()`, `insert()`, `update()` und `delete()`, um Daten zu suchen, einzufügen, zu verändern und zu löschen.

12.1.1 Mit Content Resolver auf Wörterbücher zugreifen

Während der Texteingabe macht Android dem Benutzer Wortvorschläge. Das hierfür verwendete Wörterbuch ist als Content Provider realisiert. Es lässt sich deshalb sehr einfach um eigene Einträge erweitern. Wie dies funktioniert, zeigt die App *UserDictionaryDemo*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich.

Nach dem Start werden alle Einträge des Benutzerwörterbuchs in der Sicht LogCat ausgegeben. Anschließend fügt die App das Wort »Künneth« hinzu und zeigt die Liste abermals an. Zum Schluss wird der neue Eintrag wieder entfernt. Zur Kontrolle gibt das Programm die noch vorhandenen Elemente abermals aus. Die Bildschirmausgaben sehen Sie in Abbildung 12.1.

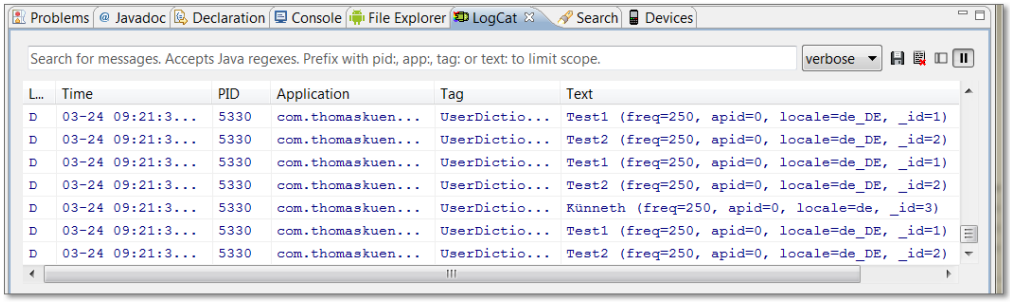


Abbildung 12.1 Die LogCat-Ausgaben der App »UserDictionaryDemo«

Tipp

Nach dem Anlegen eines virtuellen Android-Geräts ist dessen Benutzerwörterbuch leer. Sie können unter Einstellungen • Sprache & Eingabe • Persönliches Wörterbuch (bzw. Setting • Language & input • Personal dictionary auf englischsprachigen Modellen und im Emulator) Einträge hinzufügen.

Vorhandene Einträge anzeigen

Die Klasse `ContentResolver` stellt Methoden zum Suchen, Hinzufügen, Verändern und Löschen von Daten zur Verfügung. Informationen werden von Content Providern in einer Tabelle abgelegt. Jede ihrer Zeilen entspricht einem Datensatz. Operationen können sich auf eine oder mehrere Zeilen oder aber eine bestimmte Anzahl von Spalten beziehen.

Wo diese Tabelle gespeichert wird und in welchem Format dies geschieht, ist ein Implementierungsdetail des Providers und für den Nutzer der Daten unerheblich. Es kann sich also um eine SQLite-Datenbank, eine lokale Datei, einen Web-Service oder eine beliebige andere Datenquelle handeln. Der Konsument greift ausschließlich auf Methoden der Klasse `ContentResolver` zu. Wie dies aussehen kann, zeige ich Ihnen anhand der Methode `listEntries()`. Sie gibt alle Einträge des Benutzerwörterbuchs in der Sicht *LogCat* aus.

```
private void listEntries(ContentResolver cr) {
   String[] projection = { Words.WORD,
         Words.FREQUENCY, Words.APP_ID, Words.LOCALE, BaseColumns._ID };
   Cursor c = cr.query(Words.CONTENT_URI, projection, null, null, null);
   if (c == null) {
      return;
   }
   while (c.moveToNext()) {
      String word = c.getString(0);
      int freq = c.getInt(1);
      int apid = c.getInt(2);
      String locale = c.getString(3);
      long _id = c.getLong(4);
      Log.d(TAG, word + " (freq=" + freq + ", apid=" + apid + ", locale="
            + locale + ", _id=" + _id + ")");
   }
   c.close();
}
```

Listing 12.1 Auszug aus UserDictionaryDemo.java

Die Methode `query()` liefert Zeilen einer Datentabelle, die einem vorgegebenen Suchmuster entsprechen. Auf welche Tabelle zugegriffen wird und damit welcher Content Provider diese zur Verfügung stellt, steuern Sie über den ersten Parameter. Mit `Words.CONTENT_URI` adressieren Sie das Benutzerwörterbuch.

Mit dem zweiten Wert `projection`, einem Feld von Strings, legen Sie fest, welche Spalten der Tabelle das Suchergebnis enthalten sollen. `null` teilt Android mit, dass Sie alle benötigen. Die Spalte `WORD` des Benutzerwörterbuchs enthält das Wort. `FREQUENCY` gibt dessen Häufigkeit in einem Bereich zwischen 1 (selten) und 255 (gängig) an. `APP_ID` kennzeichnet die App, die einen Eintrag hinzugefügt hat. `LOCALE` kennzeichnet, in welchem Land bzw. in welcher Sprache das Wort verwendet werden soll. `_ID` schließlich ist eine für jede Tabellenzeile eindeutige Kennung.

Die beiden Parameter, die auf `projection` folgen, können die Ergebnismenge einschränken, sofern sie nicht, wie in meinem Beispiel, auf `null` gesetzt werden. Im folgenden Abschnitt zeige ich Ihnen, wie Sie diese verwenden. Der fünfte und letzte Parameter der Methode `query()` steuert die Sortierung des Ergebnisses. `null` nutzt die Standardreihenfolge des Content Providers.

Die Methode `query()` liefert ein Objekt des Typs `android.database.Cursor` oder `null`. Um eine `NullPointerException` zu vermeiden, müssen Sie deshalb vor Zugriffen auf das Objekt prüfen, ob die Referenz `null` ist. Üblicherweise werden Datenbankcursor in einer Schleife durchlaufen. Cursor repräsentieren Ergebnistabellen, verweisen aber stets auf genau eine Zeile.

Mit `moveToNext()` bewegen Sie sich zeilenweise vom Anfang zum Ende der Tabelle. Wie viele Zeilen diese enthält, ist von den Suchparametern abhängig. Aus welchen Spalten sie besteht und in welcher Reihenfolge die Spalten angeordnet sind, können Sie mit dem zweiten `query()`-Parameter (`projection`) kontrollieren. Dies ist wichtig, wenn Sie mit `get...()` auf einzelne Spalten zugreifen. Die Klasse `Cursor` kennt übrigens eine Reihe von Hilfsmethoden, mit denen Sie zum Beispiel die Nummer einer bestimmten Spalte (`getColumnIndex()`) ermitteln können.

Einträge hinzufügen

Die Methode `insert()` fügt dem Datenbestand eines Content Providers einen neuen Datensatz hinzu. Dieser wird als Objekt des Typs `android.content.ContentValues` übergeben. Durch Aufruf von `put()` wird der Wert einer bestimmten Spalte gesetzt:

```
ContentValues values = new ContentValues();
values.put(Words.APP_ID, getApplication().getApplicationInfo().uid);
values.put(Words.WORD, WORT);
```

```
values.put(Words.FREQUENCY, 250);
values.put(Words.LOCALE, Locale.GERMAN.toString());
cr.insert(Words.CONTENT_URI, values);
```

Listing 12.2 Auszug aus UserDictionaryDemo.java

Tipp

Die Klasse `android.provider.UserDictionary.Words` enthält die statische Methode `addWord()`. Auch sie ermöglicht es Ihnen, dem Benutzerwörterbuch Einträge hinzuzufügen.

Datensätze löschen

Für das Löschen von Datensätzen steht die Methode `delete()` zur Verfügung. Der erste Parameter legt wie gewohnt die Tabelle und damit den anzusprechenden Content Provider fest. Für das Benutzerwörterbuch ist dies `Words.CONTENT_URI`. Der zweite und dritte Parameter bestimmen gemeinsam, welche Zeilen der Tabelle entfernt werden.

In meinem Beispiel möchte ich alle Datensätze löschen, die das Wort »Künneth« enthalten. Der Wert der Spalte `Words.WORD` ist in so einem Fall »Künneth«. Diesen Sachverhalt formuliere ich als `Words.WORD + " is ?"`. Das Fragezeichen wird durch das erste (und einzige) Element des String-Feldes im dritten Parameter ersetzt.

```
cr.delete(Words.CONTENT_URI, Words.WORD + " is ?",
      new String[] { WORT });
```

Listing 12.3 Auszug aus UserDictionaryDemo.java

`WORT` ist eine private String-Konstante der Klasse `UserDictionaryDemo`. Der Rückgabewert von `delete()` gibt übrigens an, wie viele Zeilen gelöscht wurden.

Hinweis

Um auf die Einträge des Benutzerwörterbuchs zugreifen zu können, muss Ihre App die Berechtigung `android.permission.READ_USER_DICTIONARY` und gegebenenfalls `android.permission.WRITE_USER_DICTIONARY` anfordern. Hierzu müssen Sie wie gewohnt entsprechende Elemente in der Manifestdatei eintragen.

Wie Sie gesehen haben, ist der Zugriff auf Daten, die ein Content Provider zur Verfügung stellt, mit sehr wenigen Zeilen Quelltext möglich. Einige der Methoden von `ContentResolver` erinnern an die Klasse `android.database.sqlite.SQLiteDatabase`. Wir haben diese in Kapitel 11, »Datenbanken«, verwendet, um Datensätze der App

TKMoodley zu lesen, zu schreiben und zu modifizieren. Zwar unterscheiden sich die Methodensignaturen geringfügig, bestimmte Konzepte, wie etwa die Übergabe von Werten in Instanzen des Typs `ContentValues`, sind aber gleich.

Tatsächlich lassen sich SQLite-Datenbanken sehr elegant als Content Provider wiederverwenden. Wie das funktioniert und warum es nützlich ist, erfahren Sie in Abschnitt 12.2, »Implementierung eines eigenen Content Providers«. Zunächst möchte ich Ihnen aber noch einen weiteren in Android eingebauten Content Provider vorstellen.

12.1.2 Browser-Bookmarks

Android stellt sowohl Lesezeichen (Bookmarks) als auch URLs von besuchten Sites über einen Content Provider zur Verfügung. Wie Sie auf diesen zugreifen, zeigt die App *BookmarksDemo*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Nach dem Start gibt das Programm alle Historieneinträge und Bookmarks als Liste in der Sicht LogCat aus. Diese sehen Sie in Abbildung 12.2.

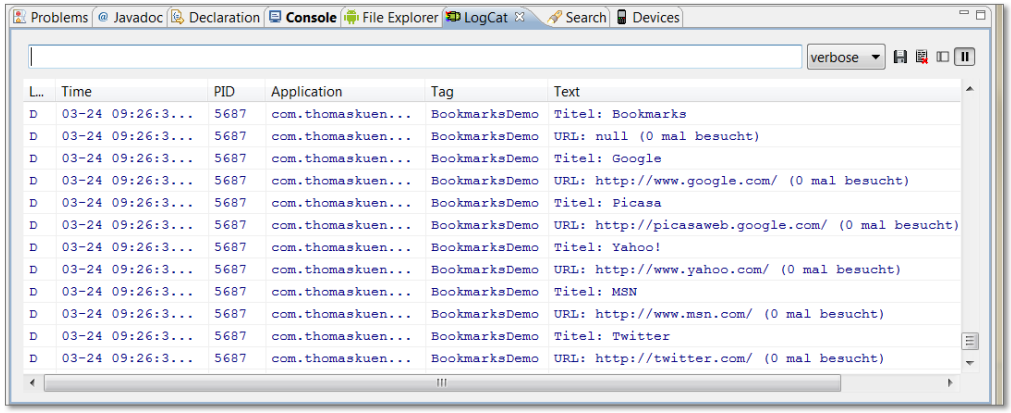


Abbildung 12.2 Ausgaben der App »BookmarksDemo«

Bookmarks auslesen

Die Implementierung erfolgt analog zum Auslesen des Benutzerwörterbuches. Zuerst müssen Sie mit `getContentResolver()` die Referenz auf ein Objekt des Typs `ContentResolver` ermitteln. Anschließend rufen Sie dessen Methode `query()` auf und iterieren über den gelieferten `Cursor`:

```
public class BookmarksDemo extends Activity {
   private static final String TAG =
      BookmarksDemo.class.getSimpleName();
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // Content Resolver ermitteln
       ContentResolver cr = getContentResolver();
       String[] projection = {
          Browser.BookmarkColumns.TITLE,
          Browser.BookmarkColumns.BOOKMARK,
          Browser.BookmarkColumns.URL,
          Browser.BookmarkColumns.VISITS };
       Cursor c = cr.query(Browser.BOOKMARKS_URI,
          projection,
          null, null, null);
       if (c != null) {
          while (c.moveToNext()) {
             Log.d(TAG, "Titel: "
                + c.getString(0)
                + (c.getInt(1) == 0 ? " (Historyeintrag)" : ""));
             Log.d(TAG, "URL: "
                + c.getString(2)
                + " (" + c.getInt(3)
                + " mal besucht)");
          }
          // Ressourcen freigeben
          c.close();
       }
    }
}
```

Listing 12.4 Auszug aus BookmarksDemo.java

Die Spalten der für Bookmarks und Historieneinträge gemeinsam genutzten Tabelle sind als Konstanten in der Klasse `Browser.BookmarkColumns` definiert. `TITLE` enthält den Namen eines Lesezeichens im Klartext, `URL` den eigentlichen Link. `VISITS` gibt die Anzahl der Seitenbesuche an. Hat `BOOKMARK` den Wert 1, wurde die Seite bereits als Lesezeichen gespeichert. 0 kennzeichnet, dass sie »nur« besucht wurde.

Lesezeichen hinzufügen und löschen

Um ein neues Lesezeichen hinzuzufügen, rufen Sie wie gewohnt die Methode `insert()` einer `ContentResolver`-Instanz auf. Mit welchen Werten Sie das zu übergebende `ContentValues`-Objekt füllen müssen, zeigt das folgende Quelltextfragment. Um es auszuprobieren, fügen Sie es in der Methode `onCreate()` der Klasse `BookmarksDemo` unterhalb der Zeile `ContentResolver cr = getContent-Resolver();` ein:

```
// ein Lesezeichen hinzufügen
ContentValues values = new ContentValues();
values.put(Browser.BookmarkColumns.TITLE, "Galileo Computing");
values.put(Browser.BookmarkColumns.BOOKMARK, 1);
values.put(Browser.BookmarkColumns.URL,
      "http://www.galileocomputing.de/");
values.put(Browser.BookmarkColumns.VISITS, 0);
cr.insert(Browser.BOOKMARKS_URI, values);
```

Listing 12.5 Quelltext zum Anlegen von Bookmarks

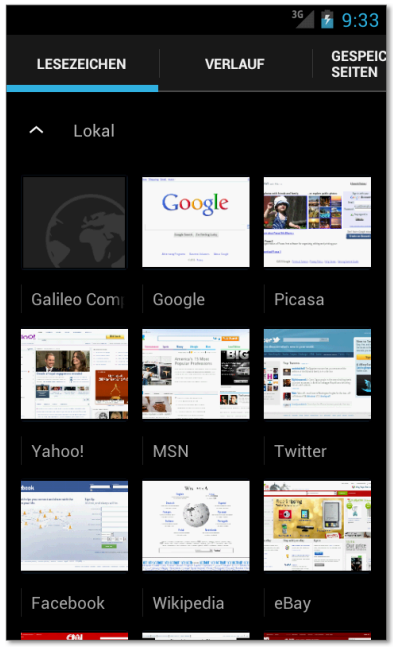


Abbildung 12.3 Bookmarks im Webbrowser

Hinweis

Um Browser-Bookmarks auslesen zu können, müssen Sie in der Manifestdatei Ihrer App die Berechtigung `com.android.browser.permission.READ_HISTORY_BOOKMARKS` anfordern. Um der Tabelle neue Zeilen hinzuzufügen oder um bestehende Zeilen zu löschen, ist zusätzlich die Berechtigung `com.android.browser.permission.WRITE_HISTORY_BOOKMARKS` erforderlich.

Wie Sie bereits wissen, enthält die Spalte `TITLE` den Namen des Bookmarks im Klartext. Die Zahl 1 bei `BOOKMARK` kennzeichnet den neuen Eintrag als Lesezeichen. Der

eigentliche Link wird `URL` zugewiesen. Und 0 bei `VISITS` zeigt an, dass die Seite noch nicht besucht wurde. Das neue Bookmark sehen Sie in Abbildung 12.3.

Bei jedem Start der App wird übrigens ein neues Lesezeichen angelegt. Wie Sie dieses wieder loswerden, möchte ich Ihnen nun zeigen. Die Klasse `ContentResolver` enthält für diesen Zweck die Methode `delete()`. Neben einem URI, der den zu verwendenden Content Provider festlegt, erwartet sie eine Auswahlbedingung sowie eine Liste mit einzusetzenden Werten. Fragezeichen innerhalb der Auswahlbedingung werden von links nach rechts durch die Elemente des dritten Parameters – ein Feld von Strings – ersetzt.

Das folgende Quelltextfragment löscht alle Lesezeichen, deren Klartextname »Galileo Computing« lautet. Der (hier nicht ausgewertete) Rückgabewert von `delete()` gibt an, wie viele Zeilen gelöscht wurden:

```
// Lesezeichen löschen
cr.delete(Browser.BOOKMARKS_URI,
      Browser.BookmarkColumns.TITLE + " is ?",
      new String[] { "Galileo Computing" });
```

Listing 12.6 Löschen von Lesezeichen

Tipp

Die Klasse `Browser` enthält die statische Methode `saveBookmark()`, mit der Sie ebenfalls Lesezeichen anlegen können.

Die Auswahlbedingung meines Beispiels erinnert Sie vielleicht an SQL. Auch wenn Content Provider und Content Resolver eine zusätzliche Datenzugriffsschicht bilden, können Sie hier tatsächlich SQL-Syntax verwenden – nur das sonst übliche Schlüsselwort `WHERE` fehlt.

12.2 Implementierung eines eigenen Content Providers

In diesem Abschnitt zeige ich Ihnen, wie Sie einen eigenen Content Provider implementieren. Wie Sie bereits wissen, können beliebige Apps auf diesen zugreifen. Deshalb sind Content Provider ideal, um Daten zu »veröffentlichen«.

12.2.1 Anpassungen an der App TKMoodley

In Kapitel 11, »Datenbanken«, haben wir das Stimmungsbarometer *TKMoodley* entwickelt, in dem der Benutzer einen von drei Smileys antippen kann. Die App legt daraufhin den ausgewählten Smiley sowie den Zeitpunkt der Erfassung in einer SQLite-

Datenbank ab. In den folgenden Abschnitten werden wir das Programm so umstellen, dass es nicht mehr direkt auf die SQLite-Datenbank, sondern auf den von uns implementierten Content Provider zugreift.

Die Klasse TKMoodley

Die Activity *TKMoodley* greift in der Implementierung im Projekt *TKMoodley_full* an insgesamt drei Stellen auf ein Objekt des Typs `TKMoodleyOpenHandler` zu. Diese Klasse leitet von `android.database.sqlite.SQLiteOpenHelper` ab und bildet die Datenzugriffsschicht der App.

Um das Programm auf die Verwendung eines Content Providers umzustellen, entfernen wir alle Referenzen auf `TKMoodleyOpenHandler` und rufen stattdessen die Ihnen bereits vertrauten Methoden von `ContentResolver` auf, in diesem Fall nur `insert()` in `imageButtonClicked()`. Das Projekt *TKMoodley_CP* enthält alle im Folgenden besprochenen Änderungen.

Wie Sie bereits wissen, erwartet `insert()` als ersten Parameter einen URI, der den zu nutzenden Content Provider referenziert. Welchen Wert die Konstante `CONTENT_URI` in diesem Fall hat, zeige ich Ihnen etwas später. Der zweite Parameter, ein Objekt des Typs `ContentValues`, enthält die einzufügenden Daten.

```
public class TKMoodley extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Benutzeroberfläche anzeigen
      setContentView(R.layout.main);

      // auf Anklicken des grünen Smileys reagieren
      final ImageButton buttonGut = (ImageButton) findViewById(R.id.gut);
      buttonGut.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            imageButtonClicked(TKMoodleyOpenHandler.MOOD_FINE);
         }
      });

      // auf Anklicken des gelben Smileys reagieren
      final ImageButton buttonOk = (ImageButton) findViewById(R.id.ok);
      buttonOk.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
            imageButtonClicked(TKMoodleyOpenHandler.MOOD_OK);
```

```
            }
        });

        // auf Anklicken des roten Smileys reagieren
        final ImageButton buttonSchlecht =
            (ImageButton) findViewById(R.id.schlecht);
        buttonSchlecht.setOnClickListener(
            new OnClickListener() {
            @Override
            public void onClick(View v) {
                imageButtonClicked(TKMoodleyOpenHandler.MOOD_BAD);
            }
        });

        // auf Anklicken der Schaltfläche Auswertung reagieren
        final Button buttonAuswertung = (Button) findViewById(R.id.auswertung);
        buttonAuswertung.setOnClickListener(
            new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(TKMoodley.this, History.class);
                startActivity(intent);
            }
        });
    }

    private void imageButtonClicked(int mood) {
        ContentValues values = new ContentValues();
        values.put(TKMoodleyOpenHandler.MOOD_MOOD, mood);
        values.put(TKMoodleyOpenHandler.MOOD_TIME,
                System.currentTimeMillis());
        getContentResolver().insert(
                TKMoodleyProvider.CONTENT_URI, values);
        // "Gespeichert"-Toast anzeigen
        Toast.makeText(this, R.string.saved, Toast.LENGTH_SHORT).show();
    }
}
```

Listing 12.7 Auszug aus TKMoodley.java

Die Activity *History* zeigt einen Verlauf der vom Benutzer erfassten Stimmungen an. Auch sie enthält Verweise auf die Klasse `TKMoodleyOpenHandler` und muss entsprechend angepasst werden, um einen Content Provider zu nutzen.

Die Klasse History

Die Historieneinträge erscheinen in einer Liste. Diese bezieht ihre Daten von einem `CursorAdapter`, den ein Objekt des Typs `android.database.Cursor` bestückt. Eine entsprechende Referenz wird in der Methode `onCreate()` ermittelt. Der Benutzer kann den Smiley-Typ bereits erfasster Einträge ändern. Dies geschieht über ein Kontextmenü. Den Aufruf von `update()` habe ich in eine eigene Methode ausgelagert. Sie zeigt die Ihnen bereits bekannte Nutzung der entsprechenden `ContentResolver`-Methode. Das Löschen findet unmittelbar in `onContextItemSelected()` statt. In beiden Fällen wird die Methode `with-AppendedPath()` aufgerufen. Content Provider gestatten den Zugriff auf individuelle Datensätze, indem an den `CONTENT_URI` eine eindeutige Kennung angehängt wird. Aus diesem Grund müssen Sie hier auch kein Auswahlkriterium festlegen und an `update()` bzw. `delete()` übergeben.

```
public class History extends ListActivity {
   // wird für die Listenansicht benötigt
   private Cursor dbCursor;
   // bildet den Cursor auf die ListView ab
   private TKMoodleyAdapter listAdapter;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Tippen und Halten öffnet Menü
      registerForContextMenu(getListView());
      // Cursor für CursorAdapter bereitstellen
      dbCursor = getContentResolver().query(
         TKMoodleyProvider.CONTENT_URI,
         null, null, null,
         TKMoodleyOpenHandler.MOOD_TIME + " DESC");
      // Activity übernimmt Verwaltung des Cursors
      startManagingCursor(dbCursor);
      // neuen ListAdapter setzen
      listAdapter = new TKMoodleyAdapter(this, dbCursor);
      setListAdapter(listAdapter);
   }

   @Override
   public void onCreateContextMenu(ContextMenu menu, View v,
         ContextMenuInfo menuInfo) {
      super.onCreateContextMenu(menu, v, menuInfo);
      // Kontextmenü entfalten
      MenuInflater inflater = getMenuInflater();
```

```
   inflater.inflate(R.menu.context_menu, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
   AdapterContextMenuInfo info =
      (AdapterContextMenuInfo) item.getMenuInfo();
   switch (item.getItemId()) {
   case R.id.menu_good:
      update(info.id, TKMoodleyOpenHandler.MOOD_FINE);
      updateList();
      return true;
   case R.id.menu_ok:
      update(info.id, TKMoodleyOpenHandler.MOOD_OK);
      updateList();
      return true;
   case R.id.menu_bad:
      update(info.id, TKMoodleyOpenHandler.MOOD_BAD);
      updateList();
      return true;
   case R.id.menu_delete:
      Uri uri = Uri.withAppendedPath(
            TKMoodleyProvider.CONTENT_URI,
            Long.toString(info.id));
      getContentResolver().delete(uri, null, null);
      updateList();
      return true;
   default:
      return super.onContextItemSelected(item);
   }
}

private void update(long id, int mood) {
   Uri uri = Uri.withAppendedPath(TKMoodleyProvider.CONTENT_URI,
         Long.toString(id));
   ContentValues values = new ContentValues();
   values.put(TKMoodleyOpenHandler.MOOD_MOOD, mood);
   getContentResolver().update(uri, values, null, null);
}
private void updateList() {
   // zunächst Cursor, dann Liste aktualisieren
   dbCursor.requery();
```

```
            listAdapter.notifyDataSetChanged();
        }
    }
```

Listing 12.8 Auszug aus History.java

Damit haben wir alle notwendigen Anpassungen abgeschlossen, um über einen Content Provider auf die TKMoodley-Datenbank zuzugreifen. Wie dieser realisiert wird, zeige ich Ihnen im Folgenden.

12.2.2 Die Klasse android.content.ContentProvider

Content Provider leiten üblicherweise von `android.content.ContentProvider` ab. Diese Klasse ist abstrakt. Sie müssen deshalb mindestens die Methoden `onCreate()`, `query()`, `insert()`, `update()`, `delete()` und `getType()` implementieren. Außerdem sollten Sie bestimmte Konstanten definieren. Besonders wichtig ist `CONTENT_URI`. Wie Sie bereits wissen, wird dieser Verweis auf einen Content Provider an sehr viele Methoden der Klasse `ContentResolver` übergeben.

Tabellenmodell und URIs

Content Provider verwalten einen oder mehrere Datentypen. Die App *TKMoodley* kennt nur die Tabelle *mood*, in der Smiley-Typen und Erfassungszeitpunkte abgelegt werden. Die beiden Spalten dieser Tabelle bilden den Datentyp `mood`. Der in diesem Abschnitt entwickelte Content Provider kennt ausschließlich diesen.

Denkbar ist aber auch, dass ein Content Provider mehrere Datentypen verwaltet. Zum Beispiel könnte er zwischen Personen, Adressen und Telefonnummern unterscheiden. Auch eine Gliederung in Untertypen ist möglich. Ein Content Provider, der Fahrzeuge verwaltet, würde wahrscheinlich zwischen Land- und Wasserfahrzeugen unterscheiden. Vielleicht fragen Sie sich nun, wie beim Aufruf einer Methode aus `ContentResolver` der gewünschte Datentopf ausgewählt wird. Üblicherweise wird ja nur ein URI übergeben.

Der URI eines Content Providers besteht aus mehreren Teilen. Er beginnt mit dem Standard-Präfix `content://`, auf das die *Authority* folgt. Sie identifiziert einen Content Provider und sollte aus einem voll qualifizierten Klassennamen bestehen, der in Kleinbuchstaben umgewandelt wurde. Wie Sie später noch sehen werden, muss die Authority in der Manifestdatei eingetragen werden.

Der nun folgende *Pfad* kennzeichnet den von mir angesprochenen Datentyp. Sofern ein Content Provider nur einen Datentyp kennt, könnte er leer bleiben. Aus Gründen der Übersichtlichkeit rate ich Ihnen aber dazu, ihn trotzdem anzugeben. Hier bietet

sich der Name der verwendeten Datenbanktabelle an. Im Falle von *TKMoodley* ist dies `mood`.

Um zwischen Untertypen zu unterscheiden, können Sie einen Pfad mit Slash (/) in mehrere Segmente teilen. URIs, die diesem Schema folgen, repräsentieren den gesamten Datenbestand eines Content Providers. Auch die Konstante `CONTENT_URI` hat dieses Format. Um einen ganz bestimmten Datensatz auszuwählen, wird dem URI noch eine eindeutige Kennung als Suffix hinzugefügt. Anwendung findet dies zum Beispiel in der Klasse `History` beim Verändern und Löschen von Einträgen.

Die Klasse TKMoodleyProvider

Wie Sie bereits wissen, leiten Content Provider üblicherweise von `android.content.ContentProvider` ab, so auch die Klasse `TKMoodleyProvider`. Sie definiert die beiden öffentlichen Konstanten `AUTHORITY` und `CONTENT_URI`. Die Konstanten `MOOD` und `MOOD_ID` hingegen sind privat. Sie werden verwendet, um bei Zugriffen auf den Content Provider zwischen dem gesamten Bestand und einzelnen Datensätzen zu unterscheiden.

Auch die statische Variable `uriMatcher` wird in diesem Zusammenhang verwendet. Sie verweist auf ein Objekt des Typs `android.content.UriMatcher`. Ein solcher Baum verknüpft Authorities und URIs mit einem Code. Dieser wird zurückgeliefert, wenn die Methode `match()` mit einem entsprechenden URI aufgerufen wird:

```
public class TKMoodleyProvider extends ContentProvider {
   public static final String AUTHORITY =
         TKMoodleyProvider.class.getName().toLowerCase();
   public static final Uri CONTENT_URI = Uri.parse("content://"
         + AUTHORITY
         + "/" + TKMoodleyOpenHandler.TABLE_NAME_MOOD);

   private TKMoodleyOpenHandler dbHelper;
   private static final int MOOD = 1;
   private static final int MOOD_ID = 2;
   private static final UriMatcher uriMatcher;
   static {
      uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
      uriMatcher.addURI(AUTHORITY,
         TKMoodleyOpenHandler.TABLE_NAME_MOOD, MOOD);

      uriMatcher.addURI(AUTHORITY,
         TKMoodleyOpenHandler.TABLE_NAME_MOOD + "/#", MOOD_ID);
```

```
    }
    ...
}
```

Listing 12.9 Auszug aus TKMoodleyProvider.java

Die Methode `onCreate()` instanziiert ein Objekt des Typs `TKMoodleyOpenHandler` und weist es der Variablen `dbHelper` zu. Der Rückgabewert `true` signalisiert, dass die Initialisierung des Content Providers erfolgreich war:

```
@Override
public boolean onCreate() {
   dbHelper = new TKMoodleyOpenHandler(getContext());
   return true;
}
```

Listing 12.10 Auszug aus TKMoodleyProvider.java

`getType()` liefert zu einem übergebenen URI einen MIME-Typ. Hierzu wird die Methode `match()` des durch `uriMatcher` referenzierten `UriMatcher` aufgerufen.

```
@Override
public String getType(Uri uri) {
   switch (uriMatcher.match(uri)) {
   case MOOD:
      // alle Einträge
      return "vnd.android.cursor.dir/vnd."
         + AUTHORITY + "/" + TKMoodleyOpenHandler.TABLE_NAME_MOOD;
   case MOOD_ID:
      // einen bestimmten Eintrag
      return "vnd.android.cursor.item/vnd."
         + AUTHORITY + "/" + TKMoodleyOpenHandler.TABLE_NAME_MOOD;
   default:
      throw new IllegalArgumentException("Unsupported URI: " + uri);
   }
}
```

Listing 12.11 Auszug aus TKMoodleyProvider.java

Die Methode `insert()` ermittelt zunächst mit `getWritableDatabase()` eine Referenz auf ein SQLite-Datenbankobjekt und fügt diesem einen neuen Datensatz hinzu:

```
@Override
public Uri insert(Uri uri, ContentValues values) {
   SQLiteDatabase db = dbHelper.getWritableDatabase();
```

```
    long rowID = db.insert(TKMoodleyOpenHandler.TABLE_NAME_MOOD,
                "", values);
    if (rowID > 0) {
       Uri result = ContentUris.withAppendedId(
                CONTENT_URI, rowID);
       getContext().getContentResolver().notifyChange(
                result, null);
       return result;
    }
    throw new SQLException("Failed to insert row into " + uri);
}
```

Listing 12.12 Auszug aus TKMoodleyProvider.java

Die Methode `query()` nutzt eine Instanz des Typs `SQLiteQueryBuilder`, anstatt direkt auf eine mit `getWritableDatabase()` ermittelte Datenbank zuzugreifen. Dies ist nötig, weil zusätzlich zu der in den beiden Parametern `selection` und `selectionArgs` übergebenen Auswahlbedingung eine Prüfung stattfinden muss, wenn nicht im gesamten Datenbestand, sondern nach einem Datensatz gesucht werden soll.


```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
      String[] selectionArgs, String sortOrder) {
   SQLiteDatabase db = dbHelper.getWritableDatabase();
   SQLiteQueryBuilder sqlBuilder = new SQLiteQueryBuilder();
   sqlBuilder.setTables(TKMoodleyOpenHandler.TABLE_NAME_MOOD);

   // Ein bestimmter Eintrag?
   if (uriMatcher.match(uri) == MOOD_ID) {
      sqlBuilder.appendWhere(TKMoodleyOpenHandler._ID + " = "
            + uri.getPathSegments().get(1));
   }
   if (sortOrder == null || sortOrder == "") {
      sortOrder = TKMoodleyOpenHandler.MOOD_TIME;
   }
   Cursor c = sqlBuilder.query(db, projection,
         selection, selectionArgs,
         null, null, sortOrder);
   // bei Änderungen benachrichtigen
   c.setNotificationUri(getContext().getContentResolver(), uri);
   return c;
}
```

Listing 12.13 Auszug aus TKMoodleyProvider.java

Die Methode `update()` prüft zunächst anhand des übergebenen URIs, ob sich Änderungen auf den gesamten Datenbestand oder nur auf einen bestimmten Datensatz beziehen sollen. In diesem Fall wird die Auswahlbedingung um ein entsprechendes Kriterium erweitert:

```
@Override
public int update(Uri uri, ContentValues values, String selection,
      String[] selectionArgs) {
   SQLiteDatabase db = dbHelper.getWritableDatabase();
   int count = 0;
   switch (uriMatcher.match(uri)) {
   case MOOD:
      count = db.update(TKMoodleyOpenHandler.TABLE_NAME_MOOD,
            values, selection, selectionArgs);
      break;
   case MOOD_ID:
      count = db.update(TKMoodleyOpenHandler.TABLE_NAME_MOOD, values,
            TKMoodleyOpenHandler._ID
            + " = "
            + uri.getPathSegments().get(1)
            + (!TextUtils.isEmpty(selection)
            ? " AND (" + selection + ')'
            : ""), selectionArgs);
      break;
   default:
      throw new IllegalArgumentException(
               "Unknown URI " + uri);
   }
   getContext().getContentResolver().notifyChange(uri, null);
   return count;
}
```

Listing 12.14 Auszug aus TKMoodleyProvider.java

Auch die Methode `delete()` prüft zunächst anhand des übergebenen URIs, ob sich Löschoperationen auf den gesamten Datenbestand oder nur auf einen bestimmten Datensatz beziehen sollen. In diesem Fall wird die Auswahlbedingung um ein entsprechendes Kriterium erweitert:

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
   SQLiteDatabase db = dbHelper.getWritableDatabase();
   int count = 0;
   switch (uriMatcher.match(uri)) {
```

```
    case MOOD:
        count = db.delete(TKMoodleyOpenHandler.TABLE_NAME_MOOD,
                selection, selectionArgs);
        break;
    case MOOD_ID:
        String id = uri.getPathSegments().get(1);
        count = db.delete(TKMoodleyOpenHandler.TABLE_NAME_MOOD,
                TKMoodleyOpenHandler._ID
                + " = "
                + id + (!TextUtils.isEmpty(selection)
                ? " AND (" + selection + ')'
                : ""), selectionArgs);
        break;
    default:
        throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```

Listing 12.15 Auszug aus TKMoodleyProvider.java

Damit ist die Implementierung des Content Providers abgeschlossen. Um ihn nutzen zu können, müssen Sie die Manifestdatei der App um folgende Zeilen erweitern:

```
<provider android:name=".TKMoodleyProvider"
    android:authorities="com.thomaskuenneth.tkmoodley.
        tkmoodleyprovider"
/>
```

Listing 12.16 Die Manifestdatei der App »TKMoodley«

Content Provider stellen tabellenartige Daten anwendungsübergreifend zur Verfügung. Die Schnittstelle wurde so konzipiert, dass sich vorhandene SQLite-Datenbanken mit minimalem Aufwand anbinden lassen. Aus welcher Quelle ein Content Provider seine Nutzdaten letztendlich bezieht, ist für den nutzenden Client vollkommen transparent. Deshalb können auch Web-Dienste oder RSS-Feeds als »Datentöpfe« dienen.

Kapitel 13
Daten sichern und wiederherstellen

Nach dem Kauf eines neuen Geräts möchte man so schnell wie möglich wieder auf bereits vorhandene Daten zugreifen. In diesem Kapitel lernen Sie, wie dies funktioniert.

Schon mit Android 2.2 (API-Level 8) hat Google Cloud-basierte Backups eingeführt. Anwendungen können wichtige Daten sichern und bei Bedarf – zum Beispiel nach dem Rücksetzen des Geräts in den Auslieferungszustand – wiederherstellen. Auch wenn der Benutzer sein Smartphone gegen ein moderneres Modell austauscht, werden Dokumente und Einstellungen automatisch auf sein neues Gerät übertragen. Daten müssen also nicht mühselig von Hand rekonstruiert werden. In diesem Kapitel zeige ich Ihnen, wie Sie Ihre Apps um diese äußerst praktische Funktion erweitern.

13.1 Grundlagen

Wie Sie gleich sehen werden, ist der Prozess des Sicherns und Wiederherstellens für den Benutzer völlig transparent und hat keine Auswirkungen auf die Funktionalität oder Bedienbarkeit Ihrer App. Während eines Backups fragt der sogenannte *Backup Manager* Ihr Programm nach zu sichernden Daten und übergibt diese einem *Backup Transport*, der die Daten an den Speicherort transferiert. Das Wiederherstellen verläuft analog, nur in umgekehrter Richtung. Der Backup Manager erhält vom Backup Transport »von irgendwoher« Daten und übergibt diese Ihrer App, die dann für die eigentliche Wiederherstellung auf dem Gerät sorgt.

Apps können die Wiederherstellung von Daten auch manuell anfordern, allerdings ist dies normalerweise nicht nötig. Denn Android führt automatisch eine Wiederherstellung durch, wenn ein Programm installiert wird, für das mit dem aktuellen Benutzer verbundene Backupdaten vorliegen. Die Hauptanwendungsfälle für eine Wiederherstellung sind das Rücksetzen eines Geräts durch den Anwender, sowie der Kauf eines neuen Geräts und das anschließende Wiederinstallieren von Apps.

Hinweis

Der in diesem Kapitel vorgestellte Sicherungsmechanismus wurde nicht für die Synchronisierung von Daten zwischen mehreren Geräten entwickelt. Er ersetzt auch nicht die in den vorherigen Kapiteln beschriebenen Techniken zum Speichern von Informationen.

13.1.1 Beteiligte Komponenten

Der *Backup Transport* ist eine clientseitige Komponente des Backup-Frameworks, die durch Gerätehersteller oder Netzanbieter angepasst werden darf. Er kann sich also von Modell zu Modell unterscheiden. Welcher Backup Transport auf einem Gerät vorhanden ist, ist für Apps aber unwichtig. Denn die Backup Manager-APIs schotten sie von der konkreten Implementierung ab. Programme kommunizieren ausschließlich über wohldefinierte Schnittstellen mit dem *Backup Manager*.

Tipp

Der Backup-Dienst ist nicht zwingend auf jedem Android-Gerät verfügbar, selbst wenn die neueste Plattform-Version darauf installiert ist. Sie können sich also nicht darauf verlassen, dass Aufrufe der im Folgenden vorgestellten APIs tatsächlich zu einer Sicherung von Daten führen. Auch die Rücksicherung ist nicht garantiert. Wenn Sie Ihre App in Google Play entsprechend bewerben, sollten Sie gegebenenfalls auf diesen Umstand hinweisen.

Wie Sie gleich noch sehen werden, haben nur Backup Manager und Backup Transport Zugriff auf zu sichernde Daten, nicht aber andere Apps. Allerdings garantiert Android nicht für die Sicherheit der Daten während des Backups, der Aufbewahrung oder der Wiederherstellung. Insofern sollten Sie genau abwägen, ob Sie auch sensible Daten wie Benutzernamen oder Passwörter auf diese Weise sichern.

Android Backup Service

Vielleicht fragen Sie sich nun, wo Sicherungsdaten eigentlich abgelegt werden, zumal ich geschrieben habe, dass auf verschiedenen Geräten unterschiedliche Backup Transports zum Einsatz kommen können. Google bietet den *Android Backup Service* an, der sehr wahrscheinlich auf den meisten Modellen (ab Android 2.2) zur Verfügung steht. Er speichert Daten in einer nicht näher spezifizierten »Cloud«.

Um diesen Backup Transport durch Ihre App nutzen zu können, müssen Sie diese unter *https://developers.google.com/android/backup/signup* registrieren. Hierzu geben Sie auf der in Abbildung 13.1 dargestellten Webseite den Paketnamen Ihres Pro-

gramms (beispielsweise `com.thomaskuenneth.backupdemo1`) ein und stimmen den Nutzungsbedingungen zu. Nach einem Klick auf REGISTER erhalten Sie auf der Folgeseite einen sogenannten Android Backup Service Key. Dieser ist in Abbildung 13.2 zu sehen.

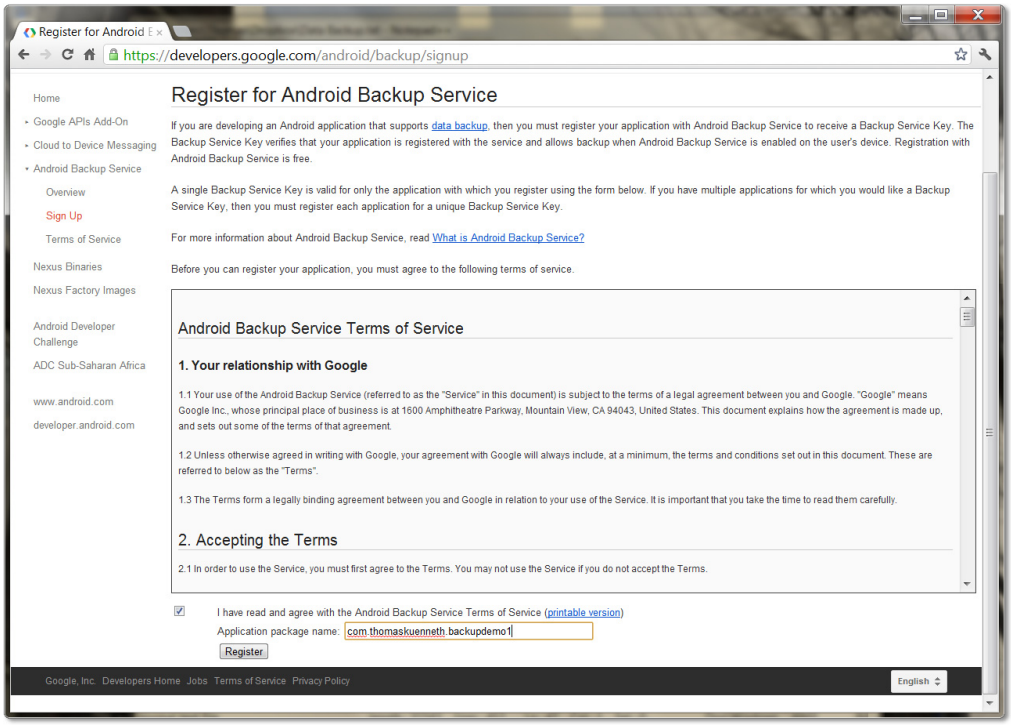


Abbildung 13.1 Registrierungsseite des Android Backup Service

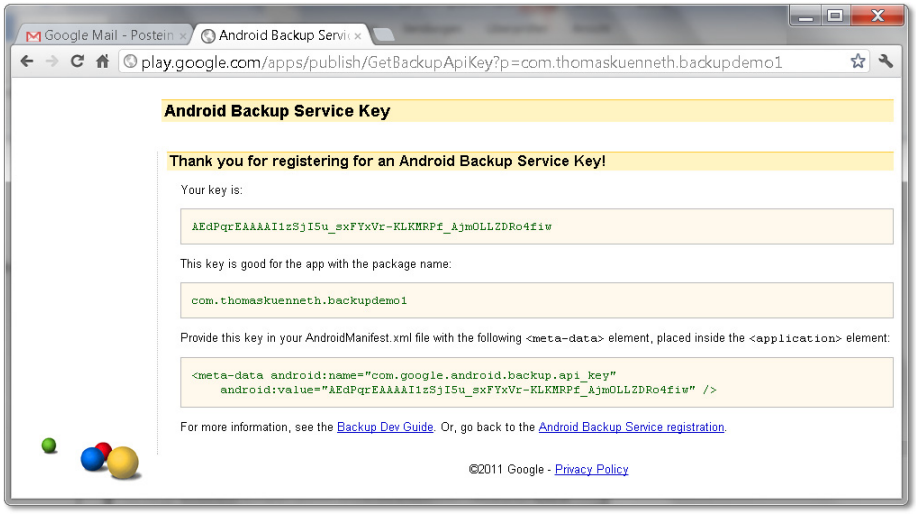


Abbildung 13.2 Webseite mit dem generierten Android Backup Service Key

Der *Android Backup Service Key* wird in der Manifestdatei Ihrer App als `<meta />`-Element innerhalb von `<application />` eingetragen.

```
<!-- https://developers.google.com/android/backup/signup -->
<meta-data android:name="com.google.android.backup.api_key"
           android:value="Ihr Android Backup Service Key" />
```

Listing 13.1 Android Backup Service Key in der Manifestdatei

Wie Sie gesehen haben, sind Schlüssel an den Paketnamen Ihrer App gebunden. Deshalb müssen Sie für jedes Programm, das Googles Android Backup Service nutzen möchte, einen eigenen Schlüssel berechnen lassen.

Hinweis

Auch Anbieter anderer Backup-Dienste können gegebenenfalls eine Registrierung verlangen. Wie Sie Ihr Programm dort registrieren, ist der entsprechenden Dokumentation zu entnehmen. Derzeit ist mir allerdings kein alternativer Anbieter eines Backupdienstes bekannt.

Backup Agent

Um Daten sichern und wiederherstellen zu können, muss Ihre App einen *Backup Agent* zur Verfügung stellen. Dieser wird später vom Backup Manager aufgerufen. Ihre Implementierung liefert die zu speichernden Daten und erhält gegebenenfalls die wiederherzustellenden. Mit Hilfe des Backup Transports übernimmt der Backup Manager die vollständige Datenübertragung an den Speicherort (im Falle des Android Backup Service in die Cloud). Ihr Backup Agent wiederum kümmert sich um Schreib- und Leseoperationen auf dem Gerät.

Backup Agents werden im Attribut `android:backupAgent` des `<application />`-Elements der Manifestdatei eingetragen. Entsprechende Klassen leiten entweder direkt von `android.app.backup.BackupAgent` oder von `android.app.backup.BackupAgentHelper` ab.

`BackupAgent` bildet die zentrale Schnittstelle für die Kommunikation Ihrer App mit dem Backup Manager. Wenn Sie unmittelbar von ihr ableiten, müssen Sie die Methoden `onBackup()` und `onRestore()` überschreiben. Oftmals ist das aber gar nicht nötig. Denn die Klasse `BackupAgentHelper` vereinfacht den Bau eines Backup Agents erheblich. Wie Sie gleich sehen werden, nutzt Ihr Programm in diesem Fall ein oder mehrere Hilfsobjekte, die ganz bestimmte Arten von Daten sichern und wiederherstellen können.

13.1.2 BackupAgentHelper

Wenn Sie Benutzereinstellungen oder eigene Dateien sichern möchten, leiten Sie am besten von `BackupAgentHelper` ab. Android kennt die beiden Implementierungen `SharedPreferencesBackupHelper` und `FileBackupHelper`. Wie Sie Erstere einsetzen, zeige ich Ihnen anhand des Beispiels *BackupDemo1*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Bitte kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich.

Die Klasse BackupDemo1Activity

Nach dem Start der App sehen Sie die in Abbildung 13.3 dargestellte Benutzeroberfläche. Sie können einen Text eingeben sowie das Häkchen eines Ankreuzfeldes nach Belieben setzen und wieder entfernen. Beim Verlassen der Activity wird der aktuelle Zustand der Bedienelemente als Benutzereinstellung gespeichert. Hierzu habe ich die Methode `onPause()` überschrieben.

```
@Override
protected void onPause() {
   super.onPause();
   SharedPreferences prefs = getSharedPreferences(NAME, MODE_PRIVATE);
   Editor editor = prefs.edit();
   editor.putString(EDITTEXT, edittext.getText().toString());
   editor.putBoolean(CHECKBOX, checkbox.isChecked());
   editor.commit();
   bm.dataChanged();
}
```

Listing 13.2 Auszug aus BackupDemo1Activity.java

Was es mit dem Aufruf `bm.dataChanged()`; auf sich hat, erkläre ich Ihnen etwas später. Die übrigen Codezeilen kennen Sie bereits aus dem Kapitel 4, »Activities und Broadcast Receiver«.

Beim (erneuten) Start der Activity werden die Benutzereinstellungen ausgelesen. Hierfür überschreiben wir die Methode `onResume()`.

```
@Override
protected void onResume() {
   super.onResume();
   SharedPreferences prefs = getSharedPreferences(NAME, MODE_PRIVATE);
   edittext.setText(prefs.getString(EDITTEXT, ""));
   checkbox.setChecked(prefs.getBoolean(CHECKBOX, false));
}
```

Listing 13.3 Auszug aus BackupDemo1Activity.java

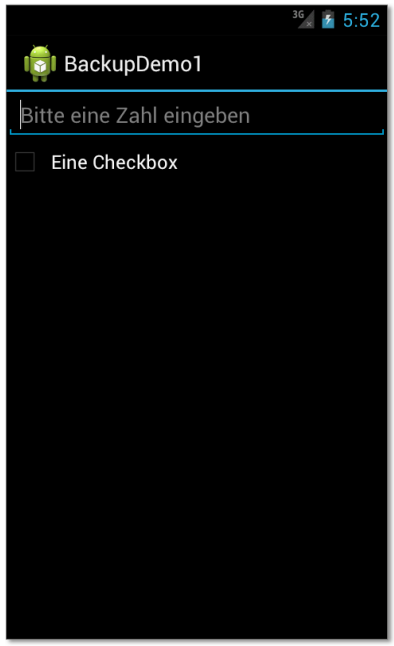


Abbildung 13.3 Die App »BackupDemo1«

Meine Implementierung greift nur auf die beiden Instanzvariablen `edittext` und `checkbox` zu. Diese werden wie üblich in der Methode `onCreate()` initialisiert.

```
public class BackupDemo1Activity extends Activity {

   public static final String NAME =
         BackupDemo1Activity.class.getSimpleName();

   private static final String EDITTEXT = "edittext";
   private static final String CHECKBOX = "checkbox";

   private BackupManager bm;

   private EditText edittext;
   private CheckBox checkbox;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      bm = new BackupManager(this);
      setContentView(R.layout.main);
      edittext = (EditText) findViewById(R.id.edittext);
```

```
      checkbox = (CheckBox) findViewById(R.id.checkbox);
   }
   ...
}
```

Listing 13.4 Auszug aus BackupDemo1Activity.java

Wie Sie sehen, ist die Variable `bm`, auf die ich in der Methode `onDestroy()` zugreife, ein Objekt des Typs `BackupManager`. Sie wird folgendermaßen initialisiert:

```
bm = new BackupManager(this);
```

Mit dem Aufruf dessen Methode `dataChanged()` teilen wir dem *Backup Manager* mit, dass sich sichernswerte Daten geändert haben, und fordern ihn auf, bei passender Gelegenheit eine Sicherung anzustoßen. Ob und wann dies geschieht, ist nicht garantiert. Sie müssen sich übrigens keine Sorgen über zu häufige Aufrufe machen. Android fasst Backup-Anfragen sinnvoll zusammen.

Vielleicht fragen Sie sich, woher der Backup Manager weiß, um welche Daten es sich handelt. Einen kleinen Hinweis habe ich Ihnen mit dem halbfett gesetzten Schlüsselwort `public` gegeben. Offenbar soll die Variable `NAME` auch von anderen Klassen verwendet werden.

Die Klasse MyBackupAgent

Wie Sie bereits wissen, werden Backup Agents in der Manifestdatei eingetragen. Im Falle von *BackupDemo1* sieht dies so aus:

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:backupAgent="MyBackupAgent"
>
```

Listing 13.5 Auszug aus der Manifestdatei von BackupDemo1

Die Implementierung der Klasse `MyBackupAgent` ist wenig spektakulär und besteht aus nur wenigen Zeilen Quelltext.

```
public class MyBackupAgent extends BackupAgentHelper {

   @Override
   public void onCreate() {
      BackupHelper h1 = new SharedPreferencesBackupHelper(this,
            BackupDemo1Activity.NAME);
```

```
        addHelper(h1.getClass().getName(), h1);
    }
}
```

Listing 13.6 Auszug aus der Klasse MyBackupAgent

`MyBackupAgent` ist ein vollständiger Backup Agent. Wenn der Backup Manager dessen Methoden `onBackup()` oder `onRestore()` aufruft, delegiert `BackupAgentHelper` die Arbeit an sogenannte *Backup Helper*. Diese werden durch das Interface `android.app.backup.BackupHelper` beschrieben. Es wird unter anderem von der Klasse `SharedPreferencesBackupHelper` implementiert. Sie kann Benutzereinstellungen sichern und wiederherstellen. Beim Instanziieren des `SharedPreferencesBackupHelper` wird der Name einer oder mehrerer solcher Dateien übergeben. Um Ihrer `BackupAgentHelper`-Implementierung einen Helper hinzuzufügen, müssen Sie in `onCreate()` die gewünschten Helper instanziieren und anschließend die Methode `addHelper()` aufrufen.

Hinweis

Für jede Datenart wird genau ein Backup Helper benötigt. Selbst wenn Ihre App also mehrere SharedPreferences verwendet, brauchen Sie trotzdem nur einen `SharedPreferencesBackupHelper`.

Im folgenden Abschnitt zeige ich Ihnen, wie Sie die Sicherungs- und Wiederherstellfunktion Ihrer App ausprobieren.

13.1.3 Den Backup Agent testen

Das Sichern und Wiederherstellen von Anwendungsdaten können Sie mit dem Emulator oder echter Hardware testen. Bitte denken Sie aber daran, dass Google diese Funktionalität erst mit Android 2.2 eingeführt hat. Sowohl ein *Android Virtual Device* (AVD) als auch gegebenenfalls das physikalische Gerät müssen also mindestens API-Level 8 unterstützen.

Als Erstes installieren Sie die Anwendung, beispielsweise *BackupDemo1*. Stellen Sie nun sicher, dass die Backup-Funktionalität aktiviert ist. Dies ist direkt im Emulator bzw. dem physikalischen Gerät oder über die Kommandozeile möglich. Sie erreichen die in Abbildung 13.4 dargestellte Einstellungsseite über Settings • Backup & reset bzw. Einstellungen • Sichern & zurücksetzen.

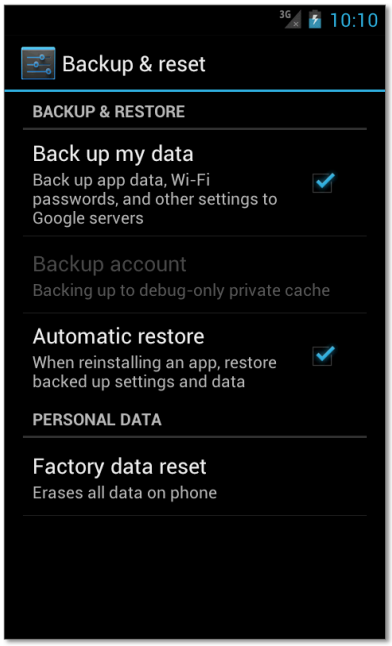


Abbildung 13.4 Backups von Anwendungsdaten ein- und ausschalten

Um die Sicherung über die Kommandozeile zu aktivieren, öffnen Sie die *Eingabeaufforderung* bzw. *Shell*. Mit der Anweisung

```
adb shell bmgr enabled
```

prüfen Sie den aktuellen Status.

```
adb shell bmgr enable true
```

schaltet die Sicherung und Wiederherstellung ein.

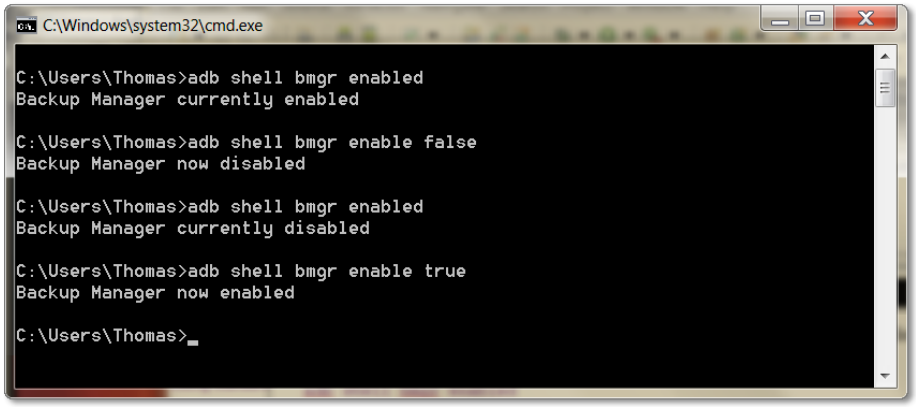


Abbildung 13.5 Mit dem Kommandozeilen-Tool bmgr arbeiten

Damit Sie die Kommandos wie in Abbildung 13.5 dargestellt eingeben können, müssen Sie die Verzeichnisse *tools* und *platform-tools* dem Standard-Suchpfad Ihres Systems hinzufügen. Weitere Infos hierzu finden Sie in Kapitel 1.

Bitte starten Sie nun *BackupDemo1*, geben in das Textfeld einen Wert ein und wechseln danach mit ZURÜCK auf den Home-Screen. Wie Sie bereits wissen, wird in der Methode `onPause()` der Klasse `BackupDemo1Activity` `dataChanged()` aufgerufen und damit eine Sicherung zu einem nicht näher bestimmten Zeitpunkt angefordert. Alternativ können Sie mit

```
adb shell bmgr backup com.thomaskuenneth.backupdemo1
```

auf der Kommandozeile ein Backup anfordern.

```
adb shell bmgr run
```

schließlich startet das Backup. Auf diese Weise werden alle ausstehenden Sicherungsanforderungen abgearbeitet. Nun können Sie die Anwendung deinstallieren. Am schnellsten geschieht dies über die Kommandozeile:

```
adb uninstall com.thomaskuenneth.backupdemo1
```

Die Aktion war erfolgreich, wenn in der *Eingabeaufforderung* bzw. *Shell* `Success` ausgegeben wird. Um den Wiederherstellungsprozess auszuprobieren, müssen Sie die App nur aus Eclipse heraus starten. Sie wird dann wie gewohnt auf dem Emulator bzw. physikalischen Gerät installiert. Dass in diesem Zusammenhang Daten rückgesichert wurden, können Sie sehr schön in der Sicht *LogCat* ablesen. Entsprechende Ausgaben sind in Abbildung 13.6 dargestellt.

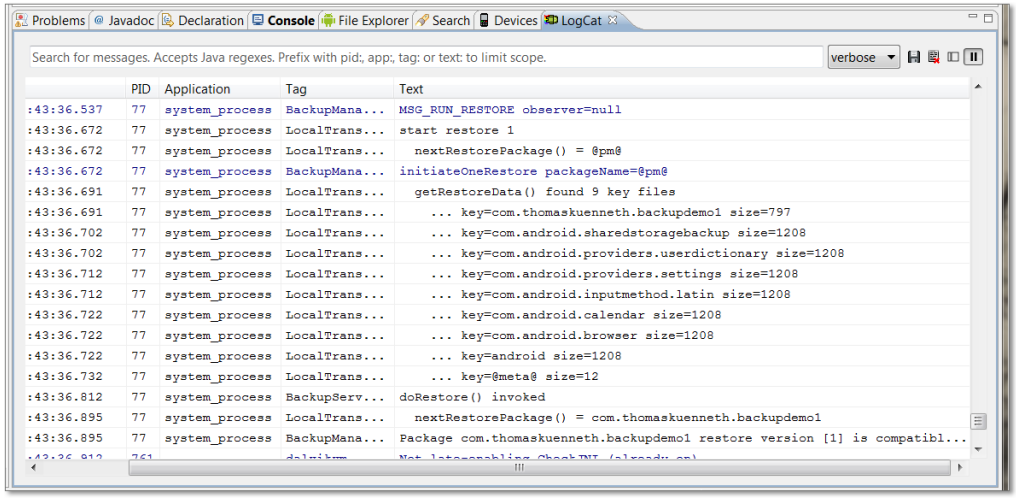


Abbildung 13.6 Bildschirmausgaben während der Rücksicherung

13.2 Eigene Dateien und Datenobjekte sichern

Nicht alle sicherungsrelevanten Daten werden als Benutzereinstellungen abgelegt. Wie Sie komplette Dateien sowie beliebige Datenstrukturen in ein Backup aufnehmen können, zeige ich Ihnen in diesem Abschnitt.

13.2.1 FileBackupHelper

Die Klasse `android.app.backup.BackupAgentHelper` nutzt für das Sichern und Wiederherstellen Hilfsobjekte, die das Interface `android.app.backup.BackupHelper` implementieren. `SharedPreferencesBackupHelper` haben Sie bereits kennen gelernt. Eine weitere Implementierung stellt die Plattform mit `FileBackupHelper` zur Verfügung.

Anhand der App *BackupDemo2* zeige ich Ihnen, wie Sie diese Klasse nutzen. Bitte kopieren und importieren Sie das gleichnamige Projekt aus dem Verzeichnis *Quelltexte* der Begleit-DVD in Ihren Eclipse-Arbeitsbereich. Die Benutzeroberfläche des Programms ist in Abbildung 13.7 zu sehen.

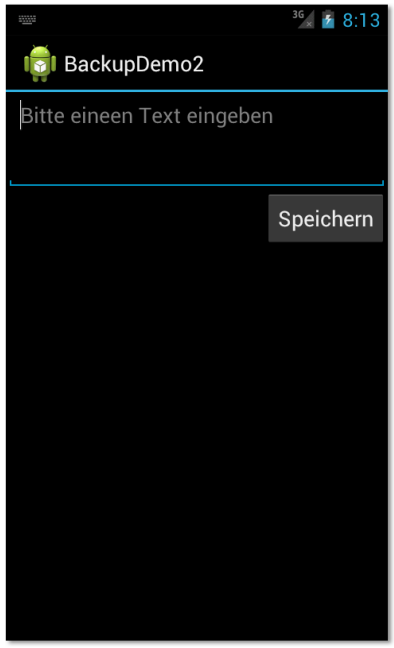


Abbildung 13.7 Die App »BackupDemo2«

Im Unterschied zu *BackupDemo1* legt die App ihre Daten nicht als `SharedPreferences` ab, sondern schreibt sie in eine Datei, nachdem der Benutzer die Schaltfläche SPEICHERN angeklickt hat.

Die Klasse BackupDemo2Activity

Bitte sehen Sie sich nun den Quelltext der Klasse `BackupDemo2Activity` an. Wie schon in *BackupDemo1* wird ein Objekt des Typs `BackupManager` instanziiert und dessen Methode `dataChanged()` aufgerufen, wenn sich Daten, die gesichert werden sollen, geändert haben. Das Schreiben und Lesen der Datei *BackupDemo2Activity.txt* findet in den Methoden `load()` und `save()` statt. Ich nutze hierfür ein Objekt des Typs `RandomAccessFile`.

```
public class BackupDemo2Activity extends Activity {

   private static final String TAG = BackupDemo2Activity.class.getSimpleName();

   // wird benötigt, um Zugriffe auf Datei zu synchronisieren
   public static final Object[] lock = new Object[0];

   // Name der Datei
   public static final String DATEINAME = "BackupDemo2Activity.txt";

   private BackupManager bm;
   private EditText edittext;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      bm = new BackupManager(this);

      setContentView(R.layout.main);
      edittext = (EditText) findViewById(R.id.edittext);

      Button button = (Button) findViewById(R.id.button);
      button.setOnClickListener(new OnClickListener() {

         @Override
         public void onClick(View v) {
            save();
            bm.dataChanged();
         }
      });
   }

   @Override
   protected void onResume() {
```

```
      super.onResume();
      load();
   }

   private void save() {
      try {
         synchronized (BackupDemo2Activity.lock) {
            File dataFile = new File(getFilesDir(), DATEINAME);
            RandomAccessFile raFile = new RandomAccessFile(dataFile, "rw");
            raFile.writeUTF(edittext.getText().toString());
            raFile.close();
         }
      } catch (IOException e) {
         Log.e(TAG, "Fehler beim Schreiben der Datei " + DATEINAME);
      }
   }

   private void load() {
      try {
         synchronized (BackupDemo2Activity.lock) {
            File dataFile = new File(getFilesDir(), DATEINAME);
            RandomAccessFile raFile = new RandomAccessFile(dataFile, "r");
            String text = raFile.readUTF();
            edittext.setText(text);
            raFile.close();
         }
      } catch (IOException e) {
         Log.e(TAG, "Fehler beim Lesen der Datei " + DATEINAME);
      }
   }
}
```

Listing 13.7 Die Klasse BackupDemo2Activity

Ist Ihnen aufgefallen, dass ich den Zugriff auf die Datei *BackupDemo2Activity.txt* in einen `synchronized`-Block gepackt habe? Warum dies notwendig ist, zeige ich Ihnen gleich.

Tipp

Ein Feld der Länge 0 ist leichtgewichtiger als ein »normales« Objekt. Es eignet sich deshalb besonders gut als Lock.

Die Klasse FileBackupAgent

Wie Sie aus den vorangehenden Abschnitten wissen, überschreiben Klassen, die von `BackupAgentHelper` ableiten, üblicherweise die Methode `onCreate()`. Sie stammt übrigens aus `BackupAgent`, der Elternklasse von `BackupAgentHelper`. In `onCreate()` werden Hilfsklassen instanziiert und mittels `addHelper()` registriert.

```
public class FileBackupAgent extends BackupAgentHelper {

   // Schlüssel zum Identifizieren der Backup-Daten
   static final String BACKUP_KEY = FileBackupAgent.class.getSimpleName();

   @Override
   public void onCreate() {
      FileBackupHelper helper =
            new FileBackupHelper(this, BackupDemo2Activity.DATEINAME);
      addHelper(BACKUP_KEY, helper);
   }

   @Override
   public void onBackup(ParcelFileDescriptor oldState,
                        BackupDataOutput data,
                        ParcelFileDescriptor newState)
               throws IOException {
      synchronized (BackupDemo2Activity.lock) {
         super.onBackup(oldState, data, newState);
     }
   }

   @Override
   public void onRestore(BackupDataInput data,
                         int appVersionCode,
                         ParcelFileDescriptor newState)
               throws IOException {
      synchronized (BackupDemo2Activity.lock) {
         super.onRestore(data, appVersionCode, newState);
      }
   }
}
```

Listing 13.8 Die Klasse FileBackupAgent

Meine Implementierung `FileBackupAgent` überschreibt zusätzlich die Methoden `onBackup()` und `onRestore()` von `BackupAgent`. Die `synchronized`-Anweisung stellt

sicher, dass während der Sicherung bzw. Rücksicherung kein anderer Teil der App auf die Datei *BackupDemo2Activity.txt* zugreifen kann.

Die Backup-Funktionalität von Android erlaubt also auch das Speichern und Wiederherstellen von physikalischen Dateien. Der Aufwand hierfür ist nur geringfügig größer als beim Backup von Benutzereinstellungen.

13.2.2 Von BackupAgent ableiten

Stellen Sie sich vor, wichtige Informationen Ihrer App sind über mehrere Dateien verteilt. Statt diese einzeln mit dem *Android Backup Service* zu sichern, bietet es sich an, die relevanten Inhalte in einer Datenstruktur zu bündeln und diese dem Backup Manager zu übergeben.

Wie Sie hierzu vorgehen, zeige ich Ihnen anhand des Programms *BackupDemo3*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Bitte kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich. Die App ist in Abbildung 13.8 zu sehen. Wenn Sie die Schaltfläche Übernehmen anklicken, werden die Inhalte der beiden Eingabefelder lokal auf dem Gerät abgelegt. Anschließend fordert das Programm wie gewohnt mit `dataChanged()` ein Backup an und beendet sich danach mit `finish()`. Bitte werfen Sie nun einen Blick auf den folgenden Auszug aus der Klasse `BackupDemo3`.

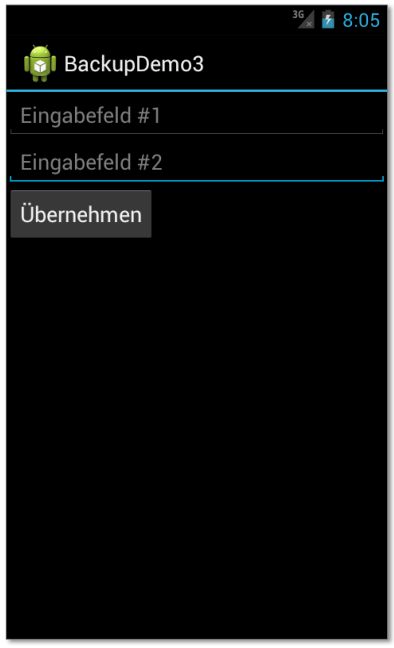


Abbildung 13.8 Die App »BackupDemo2«

Die meisten Anweisungen und Konzepte werden Ihnen sicher aus den vorangehenden Abschnitten vertraut sein. Aber vielleicht fragen Sie sich, was es mit den Operationen auf Objekten des Typs `JSONObject` auf sich hat.

Die JSON API

Die *JavaScript Object Notation*, kurz *JSON*, ist ein kompaktes Datenformat, das sowohl für Mensch als auch Maschine einfach lesbar ist. Es wird gerne für den Datenaustausch zwischen (Web-)Anwendungen eingesetzt. Android gestattet den einfachen Zugriff auf solche Datenstrukturen.

```
public class BackupDemo3Activity extends Activity {

   public static final Object[] LOCK = new Object[0];
   public static final String INPUT1 = "input1";
   public static final String INPUT2 = "input2";

   private static final String TAG =
         BackupDemo3Activity.class.getSimpleName();

   private BackupManager bm;

   private EditText input1, input2;
   private Button save;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      bm = new BackupManager(this);

      setContentView(R.layout.main);
      input1 = (EditText) findViewById(R.id.input1);
      input2 = (EditText) findViewById(R.id.input2);
      save = (Button) findViewById(R.id.save);

      save.setOnClickListener(new OnClickListener() {

         @Override
         public void onClick(View v) {
            // als JSON-Objekt in Datei schreiben
            JSONObject object = new JSONObject();
            try {
                  object.put(INPUT1, input1.getText().toString());
                  object.put(INPUT2, input2.getText().toString());
```

```
                FileUtilities.save(BackupDemo3Activity.this, object);
                  // Sicherung anfordern
                bm.dataChanged();
             } catch (JSONException e) {
                Log.e(TAG, "Problem mit der JSON API", e);
             }
             // Activity beenden
             finish();
          }
       });

       JSONObject object = FileUtilities.load(this);
       if (object != null) {
          try {
             input1.setText(object.getString(INPUT1));
             input2.setText(object.getString(INPUT2));
          } catch (JSONException e) {
             Log.e(TAG, "Problem mit der JSON API", e);
          }
       }
    }
}
```

Listing 13.9 Auszug aus BackupDemo3.java

Die Implementierung der Methoden `load()` und `save()` habe ich in die Klasse `FileUtilities` ausgelagert. Ihr Quelltext ist im Folgenden zu sehen.

```
public class FileUtilities {

   private static final String TAG =
         FileUtilities.class.getSimpleName();
   private static final String DATEINAME = TAG + ".json";

   public static File getFile(ContextWrapper cw) {
      File dataFile = new File(cw.getFilesDir(), DATEINAME);
      return dataFile;
   }

   public static void save(ContextWrapper cw,
                           JSONObject object) {
      try {
         synchronized (BackupDemo3Activity.LOCK) {
            File f = new File(cw.getFilesDir(), DATEINAME);
```

```
            RandomAccessFile raFile = new RandomAccessFile(f, "rw");
            raFile.writeUTF(object.toString());
            raFile.close();
         }
      } catch (IOException e) {
         Log.e(TAG, "Fehler beim Schreiben der Datei " + DATEINAME);
      }
   }

   public static JSONObject load(ContextWrapper cw) {
      JSONObject object = null;
      synchronized (BackupDemo3Activity.LOCK) {
         File dataFile = new File(cw.getFilesDir(), DATEINAME);
         RandomAccessFile raFile = null;
         try {
            raFile = new RandomAccessFile(dataFile, "r");
            String data = raFile.readUTF();
            object = new JSONObject(data);
         } catch (JSONException e) {
            Log.e(TAG, "Problem mit der JSON API", e);
         } catch (FileNotFoundException e) {
            Log.e(TAG, DATEINAME + " nicht gefunden", e);
         } catch (IOException e) {
            Log.e(TAG, "Fehler beim Lesen der Datei " + DATEINAME, e);
         } finally {
            try {
               if (raFile != null) {
                  raFile.close();
               }
            } catch (IOException e) {
            }
         }
      }
      return object;
   }
}
```

Listing 13.10 Auszug aus der Klasse FileUtilities

JSON-Daten sind ein- oder mehrzeilige Zeichenketten. Das Speichern reduziert sich deshalb auf das Ablegen in einer Datei. Neben `toString()` kennt die Klasse `JSONObject` eine Variante dieser Methode, der Sie die Anzahl der Leerzeichen je Schachtelungstiefe übergeben. Dies erleichtert die Fehlersuche. Die JSON API bietet auch einen

komfortablen Weg, um JSON-Objekte zu erzeugen. Wie Sie in der Methode `load()` sehen, brauchen Sie nur den Dateiinhalt in Form eines Strings an den Konstruktor zu übergeben.

Die Klasse StructureBackupAgent

Sicherlich sind Sie schon neugierig, wie eine eigene, von `BackupAgent` abgeleitete Implementierung aussieht und wie umfangreich sie ist. Bitte sehen Sie sich hierzu zunächst den Quelltext der Klasse `StructureBackupAgent` an.

```
public class StructureBackupAgent extends BackupAgent {

   private static final String TAG
         = StructureBackupAgent.class.getSimpleName();

   private File mDataFile;

   @Override
   public void onCreate() {
      mDataFile = FileUtilities.getFile(this);
   }

   @Override
   public void onBackup(ParcelFileDescriptor oldState,
                        BackupDataOutput data,
                        ParcelFileDescriptor newState)
         throws IOException {
      // Änderungsdatum der lokalen Datei ermitteln
      long fileModified = mDataFile.lastModified();
      // Dateidatum während des letzten Backups ermitteln
      long stateModified = 0L;
      try {
         if (oldState != null) {
            FileInputStream instream = new FileInputStream(
                  oldState.getFileDescriptor());
            DataInputStream in = new DataInputStream(instream);
            try {
               stateModified = in.readLong();
            } catch (IOException e) {
               // wir ignorieren diese Ausnahme und erzwingen
               // damit ein Backup
            }
         }
         if ((stateModified != fileModified) || (stateModified == 0l)) {
```

```
            // lokale Datei ist vorhanden und
            // wurde geändert, also Backup durchführen
            JSONObject object = FileUtilities.load(this);
            try {
               byte[] b1 =
                  object.getString(BackupDemo3Activity.INPUT1).getBytes();
               int l1 = b1.length;
               byte[] b2 =
                  object.getString(BackupDemo3Activity.INPUT2).getBytes();
               int l2 = b2.length;
               // Daten an Backup Manager senden
               data.writeEntityHeader(BackupDemo3Activity.INPUT1, l1);
               data.writeEntityData(b1, l1);
               data.writeEntityHeader(BackupDemo3Activity.INPUT2, l2);
               data.writeEntityData(b2, l2);
            } catch (JSONException e) {
               Log.e(TAG, "Problem mit der JSON API", e);
            }
         } else {
            // Datei hat sich nicht geändert - kein Backup
            // ggf. Aufräumaktionen durchführen
         }
      } catch (IOException e) {
         Log.e(TAG, "Problem beim Zugriff auf Dateien", e);
         // Google rät, trotzdem ein Backup zu machen;
         // wurde hier nicht ausprogrammiert
      } finally {
         FileOutputStream outstream = new FileOutputStream(
               newState.getFileDescriptor());
         DataOutputStream out = new DataOutputStream(outstream);
         out.writeLong(fileModified);
      }
   }

   @Override
   public void onRestore(BackupDataInput data,
                         int appVersionCode,
                         ParcelFileDescriptor newState)
         throws IOException {
      Log.d(TAG, "App-Version, mit der Backup erstellt wurde: "
               + appVersionCode);
      Log.d(TAG, "Aktuelle Version der App: "
               + getVersionCode());
      JSONObject object = new JSONObject();
```

```
        while (data.readNextHeader()) {
            String key = data.getKey();
            int dataSize = data.getDataSize();
            // Backup-Datensatz lesen
            byte[] dataBuf = new byte[dataSize];
            data.readEntityData(dataBuf, 0, dataSize);
            String s = new String(dataBuf);
            // Ist uns der Schlüssel bekannt?
            try {
               if (BackupDemo3Activity.INPUT1.equals(key)) {
                  object.put(BackupDemo3Activity.INPUT1, s);
               } else if (BackupDemo3Activity.INPUT2.equals(key)) {
                  object.put(BackupDemo3Activity.INPUT2, s);
               }
            } catch (JSONException e) {
               Log.e(TAG, "Problem mit der JSON API", e);
            }
        }
        FileUtilities.save(this, object);
        // Backup-Zeitstempel aktualisieren
        FileOutputStream outstream = new FileOutputStream(
              newState.getFileDescriptor());
        DataOutputStream out = new DataOutputStream(outstream);
        long fileModified = mDataFile.lastModified();
        out.writeLong(fileModified);
    }

    private int getVersionCode() {
        PackageInfo info;
        try {
            String name = getPackageName();
            info = getPackageManager().getPackageInfo(name, 0);
        } catch (NameNotFoundException nnfe) {
            info = null;
        }
        int versionCode = 0;
        if (info != null) {
            versionCode = info.versionCode;
        }
        return versionCode;
    }
}
```

Listing 13.11 Die Klasse StructureBackupAgent

Eine sehr wichtige Methode der Klasse `BackupAgent` ist `onBackup()`. Sie wird vom *Backup Manager* aufgerufen, wenn eine Sicherung durchgeführt werden soll. Wie Sie bereits wissen, geschieht dies »irgendwann«, nachdem Ihre App mit `dataChanged()` eine solche angefordert hat.

`onBackup()` erhält hierzu drei Parameter. `oldState` verweist auf den letzten Sicherungsstatus, der von Ihrem Programm geliefert wurde. Hierbei handelt es sich nicht um die in der Cloud liegenden Daten, sondern um eine lokale Repräsentation. Sie verwenden diese, um zu überprüfen, ob sich Daten seit dem letzten Backup geändert haben. Mit dem zweiten Parameter, `data`, übergeben Sie Ihre Daten an den Backup Manager. `newState` schließlich wird verwendet, um die gesicherten Daten zu charakterisieren. Hierbei kann es sich um Hashwerte oder Zeitstempel handeln. Beim nächsten Aufruf von `onBackup()` wird `newState` als `oldState` übergeben.

Tipp

Welche Informationen Sie im Sicherungsstatus ablegen, hängt letztlich von den Daten ab, die Ihre App verarbeitet. Wenn Sie Dateien sichern möchten, reicht oftmals deren Änderungsdatum, gegebenenfalls in Verbindung mit der Länge in Byte. Bei größeren Datenmengen kann es sich lohnen, anhand des Sicherungsstatus zwischen Voll- und Teilbackups zu unterscheiden.

Daten werden als sogenannte *Entities* nach `BackupDataOutput` geschrieben. Eine *Entity* ist eine flache binäre Datenstruktur, die mit einem eindeutigen Schlüssel identifiziert wird. Die zu schreibenden Daten sind also Schlüssel-Wert-Paare. Um dem Backup-Datensatz Entitäten hinzuzufügen, rufen Sie zunächst `writeEntityHeader()` auf und übergeben einen eindeutigen Schlüssel für die Daten sowie deren Länge in Bytes. `writeEntityData()` übergibt die Daten an den Backup Manager. Diese Schritte müssen Sie für alle zu sichernden Datenbereiche durchführen. Auf wie viele Entitäten Sie Ihre Daten verteilen, ist Ihnen überlassen.

Hinweis

Bitte beachten Sie, dass Sie auch hier mit `synchronized`-Anweisungen den exklusiven Zugriff auf Dateien sicherstellen müssen.

Während einer Wiederherstellung ruft der Backup Manager die Methode `onRestore()` auf und übergibt Ihrer App die rückzusichernden Daten. Wie Sie bereits wissen, geschieht dies automaisch, wenn während der Installation entdeckt wird, dass gültige Backupdaten vorhanden sind. Mit `requestRestore()` können Sie gegebenenfalls eine manuelle Wiederherstellung anfordern. Ob und wann diese tatsächlich durchgeführt wird, ist nicht vorhersagbar.

Auch `onRestore()` erhält drei Parameter: `data`, `appVersionCode` und `newState`. Eine Implementierung ruft üblicherweise so lange `readNextHeader()` auf, wie Daten vorliegen. Für jede gefundene Entität wird mit `getKey()` deren Schlüssel ermittelt. Diesen vergleichen Sie mit denen Ihrer App und speichern die auf diese Weise geladenen Daten lokal auf dem Gerät.

Anhand des Parameters `appVersionCode` können Sie sehen, mit welcher Version Ihres Programms die wiederherzustellenden Daten geschrieben wurden. Meine Methode `getVersionCode()` ermittelt die aktuelle App-Version. Wie Sie wissen, wird diese im Attribut `android:versionCode` der Manifestdatei hinterlegt. Durch Vergleich der beiden Werte lassen sich gegebenenfalls Teile von einer Wiederherstellung ausschließen.

In Abbildung 13.9 sehen Sie die Ausgabe von zwei unterschiedlichen Versionsnummern in der Sicht *LogCat*. Hierzu habe ich nach dem Anklicken der Schaltfläche Übernehmen mit den Ihnen bereits bekannten Kommandos `bmgr backup com.thomaskuenneth.backupdemo3` und `bmgr run` eine Sicherung durchgeführt. Nach der Deinstallation der App habe ich die Versionsnummer auf `123` gesetzt und danach das Programm auf den Emulator übertragen.

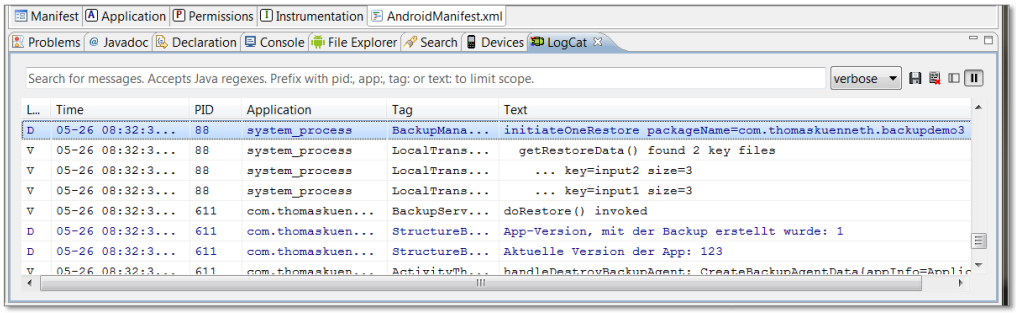


Abbildung 13.9 Ausgabe von Versionsnummern in der Sicht LogCat

Weitere Hinweise zum Umgang mit der Backup API finden Sie im Dokument *Data Backup* von Googles Entwicklerdokumentation.[1]

1 *http://developer.android.com/guide/topics/data/backup.html*

TEIL V

Organizer und Multimedia

Kapitel 14
Audio

Android bietet vielfältige Möglichkeiten, Audio aufzunehmen und wiederzugeben. Die Nutzung der entsprechenden Programmierschnittstellen zeige ich Ihnen anhand einer Diktiergerät-App.

In diesem Kapitel möchte ich Sie mit den beeindruckenden Audiofähigkeiten der Android-Plattform vertraut machen. Das System kann nicht nur die unterschiedlichsten Formate abspielen, sondern den Audiostrom auch mit akustischen Effekten versehen. Ein weiterer spannender Aspekt ist die Sprachsynthese, also die Umwandlung von geschriebener in gesprochene Sprache. Wie Sie mit Ihrer App Texte vorlesen lassen, zeige ich Ihnen am Ende dieses Kapitels.

14.1 Rasender Reporter – ein Diktiergerät als App

Diktiergeräte sind eine ausgesprochen praktische Erfindung – Aufnahmeknopf drücken und Mikrofon in Richtung der Tonquelle halten. Da sich an so einem Beispiel sehr schön zeigen lässt, wie Sie mit Android Audiosignale aufzeichnen und später wieder abspielen können, habe ich die App *RR* (die beiden Buchstaben stehen für »Rasender Reporter«) entwickelt. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich.

14.1.1 Struktur der App

Nach dem ersten Start sehen Sie einen fast leeren Bildschirm. Am unteren Rand befindet sich die Schaltfläche AUFNEHMEN. Ein Klick startet bzw. stoppt die Aufnahme. Haben Sie auf diese Weise ein Signal gespeichert, erscheint die korrespondierende Datei in einer Liste. Sie sehen sie in Abbildung 14.1. Um eine Aufnahme abzuspielen, klicken Sie einfach den entsprechenden Eintrag an. Sie können die Wiedergabe durch Anklicken von BEENDEN jederzeit abbrechen.

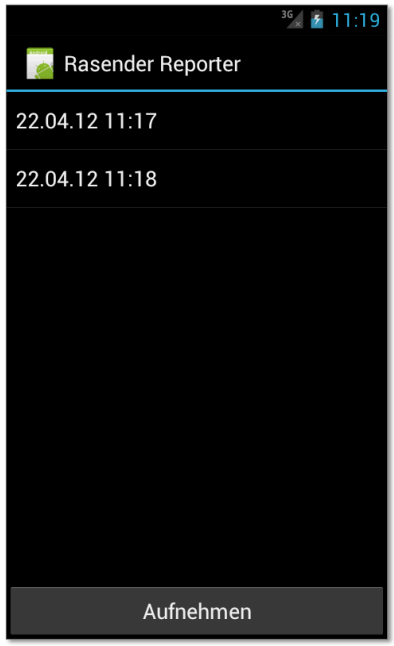


Abbildung 14.1 Die Benutzeroberfläche der App »RR«

Die App besteht aus der Activity `RR` sowie den beiden Hilfsklassen `RRFile` und `RRListAdapter`. `RR` stellt nicht nur die Benutzeroberfläche dar, sondern übernimmt auch das Aufnehmen und Abspielen. In ihr wird also die eigentliche Programmlogik implementiert. Ausführliche Informationen zu dieser Klasse finden Sie in Abschnitt 14.1.2, »Audio aufnehmen und abspielen«.

`RRListAdapter` wird benötigt, um die aufgenommenen Audioschnipsel in einer Liste anzuzeigen. `RRFile` ist eine weitere technische Hilfsklasse, die ebenfalls die Ausgabe in der Liste vereinfacht.

Die Klasse RRFile

Die Klasse `RRFile` hält die Daten, die in der eben angesprochenen Liste angezeigt werden. Hierbei handelt es sich um die Namen der gespeicherten Dateien. Sie liegen als Objekte des Typs `RRFile` vor. Diese Klasse leitet von `java.io.File` ab und überschreibt deren Methode `toString()`. Der Name einer Datei ohne Erweiterung (wie Sie später noch sehen werden, lautet diese *.3gp*) wird als Zahl interpretiert und in ein `java.util.Date`-Objekt umgewandelt. Dieses Datum wiederum wird in Klartext als Ergebnis zurückgegeben.

```
@Override
public String toString() {
   String result = getName().toLowerCase();
```

```
   result = result.substring(0, result.indexOf(EXT_3GP));
   try {
      Date d = new Date(Long.parseLong(result));
      result = DateFormat.getInstance().format(d);
   } catch (Throwable tr) {
   }
   return result;
}
```

Listing 14.1 Auszug aus RRFile.java

Die Klasse RRListAdapter

Vielleicht fragen Sie sich nun, warum das Überschreiben von `toString()` hilfreich sein sollte. Lassen Sie uns hierzu einen Blick auf `RRListAdapter` werfen. Diese Klasse leitet von `android.widget.ArrayAdapter` ab und speichert die Daten, die von einer `ListView` angezeigt werden, in einem zur Laufzeit veränderbaren Feld ab.

Das Aussehen eines Listenelements wird immer durch die Methode `getView()` bestimmt. Sie ist im Basisinterface `android.widget.Adapter` definiert. Die Implementierung in `ArrayAdapter` entfaltet eine Layoutdatei, die dem Konstruktor übergeben wurde. Das Layout besteht üblicherweise aus einem einzigen Element – einer `TextView`. Der anzuzeigende Text ergibt sich aus dem Aufruf der Methode (Sie ahnen es sicher) `toString()` des mit dem Listenelement verknüpften Werts.

Die private Methode `findAndAddFiles()` ermittelt alle Dateien, die in einem bestimmten Basisverzeichnis liegen, und fügt diese durch Aufruf von `add()` dem internen Feld als `RRFile`-Instanzen hinzu:

```
public class RRListAdapter extends ArrayAdapter<File> {
   public RRListAdapter(Context context) {
      super(context, android.R.layout.simple_list_item_1);
      // vorhandene Dateien suchen und hinzufügen
      findAndAddFiles();
   }

   private void findAndAddFiles() {
      File dir = RR.getBaseDir();
      File[] files = dir.listFiles(new FilenameFilter() {
         @Override
         public boolean accept(File dir, String filename) {
            if (!filename.toLowerCase().endsWith(RRFile.EXT_3GP)) {
               return false;
            }
            File f = new File(dir, filename);
```

```
            return f.canRead() && !f.isDirectory();
         }
      });

      if (files != null) {
         for (File f : files) {
            add(new RRFile(f.getParentFile(), f.getName()));
         }
      }
   }
}
```

Listing 14.2 Auszug aus RRListAdapter.java

Tipp

Ist Ihnen die Nutzung von `FilenameFilter` aufgefallen? `File`-Objekte kennen die Methode `listFiles()`, um den Inhalt eines Verzeichnisses zu bestimmen. Sie können ihr einen Filter übergeben und damit kontrollieren, welche Elemente in das Ergebnis (`File[]`) übernommen werden.

14.1.2 Audio aufnehmen und abspielen

In diesem Abschnitt werden wir uns die Klasse `RR` näher ansehen. Diese Activity stellt die Benutzeroberfläche des Rasenden Reporters dar und kümmert sich um das Aufnehmen und Wiedergeben von Audiosignalen. `RR` kennt drei Zustände, die das `enum MODE` durch die Werte `WAITING`, `RECORDING` und `PLAYING` repräsentiert. Das Programm zeichnet also entweder auf, spielt ab oder wartet auf Aktionen des Benutzers.

Der aktuelle Zustand wird in der Instanzvariablen `mode` abgelegt und steuert sowohl die Beschriftung als auch das Verhalten der Schaltfläche am unteren Bildschirmrand. Was beim Anklicken passiert, können Sie in der `if`-Abfrage in der Methode `onClick()` des entsprechenden `OnClickListener` sehen. `updateButtonText()` setzt den Text des Buttons.

Außer der Methode `onCreate()` habe ich auch `onPause()` überschrieben. Sie wird immer aufgerufen, wenn die Arbeit mit einer Activity unterbrochen wird. In so einem Fall sollte die Wiedergabe oder die Aufnahme beendet werden. Was die privaten Methoden `releasePlayer()` und `releaseRecorder()` tun, zeige ich Ihnen etwas später.

`getBaseDir()` liefert das Verzeichnis, in dem Aufnahmen abgelegt werden. Ich erzeuge auf dem externen Medium den Ordner *.RR*, damit die Audiodateien des

Rasenden Reporters nicht automatisch in die Mediendatenbank übernommen werden. Wenn Sie eine Indizierung wünschen, entfernen Sie einfach den führenden Punkt des Verzeichnisnamens.

Tipp

Wie Sie aus Kapitel 10, »Das Android-Dateisystem«, wissen, können Sie mit `Environment.getExternalStorageState()` den Status des externen Speichermediums abfragen. Um Ihre App möglichst robust zu machen, sollten Sie vor Zugriffen prüfen, ob Schreib- bzw. Lesevorgänge aktuell möglich sind.

```
public class RR extends Activity {
   private static final String TAG = RR.class.getSimpleName();

   // wird für die Schaltfläche benötigt
   private static enum MODE {
      WAITING, RECORDING, PLAYING
   };
   private MODE mode;

   // Bedienelemente der App
   private RRListAdapter listAdapter;
   private Button b;

   // Datei mit der aktuellen Aufnahme
   private File currentFile;

   private MediaPlayer player;
   private MediaRecorder recorder;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      // ListView initialisieren
      final ListView lv = (ListView) findViewById(R.id.listview);
      listAdapter = new RRListAdapter(this);
      lv.setAdapter(listAdapter);
      lv.setOnItemClickListener(new OnItemClickListener() {
         @Override
         public void onItemClick(AdapterView<?> parent,
               View view, int position, long id) {
```

```
            // Datei wiedergeben
            File f = listAdapter.getItem(position);
            playAudioFile(f.getAbsolutePath());
        }
    });

    // Schaltfläche Aufnehmen/Beenden initialisieren
    b = (Button) findViewById(R.id.button);
    b.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (mode == MODE.WAITING) {
                currentFile = recordToFile();
            } else if (mode == MODE.RECORDING) {
                // die Aufnahme stoppen
                recorder.stop();
                releaseRecorder();
                listAdapter.add(currentFile);
                currentFile = null;
                mode = MODE.WAITING;
                updateButtonText();
            } else if (mode == MODE.PLAYING) {
                player.stop();
                releasePlayer();
                mode = MODE.WAITING;
                updateButtonText();
            }
        }
    });
    currentFile = null;
    mode = MODE.WAITING;
    player = null;
    recorder = null;
    updateButtonText();
}

@Override
protected void onPause() {
    super.onPause();
    releasePlayer();
    releaseRecorder();
}
```

```
   private void updateButtonText() {
      b.setText(getString((mode != MODE.WAITING)
            ? R.string.finish
            : R.string.record));
   }

   // Hier fehlen noch vier Methoden.
   ...

   public static File getBaseDir() {
      File dir = new File(
         Environment.getExternalStorageDirectory(), ".RR");
      // ggf. Verzeichnisse anlegen
      dir.mkdirs();
      return dir;
   }
}
```

Listing 14.3 Auszug aus RR.java

Audiodateien wiedergeben

Das Anklicken eines Listenelements startet die Wiedergabe einer Audiodatei. Hierzu wird die private Methode `playAudioFile()` aufgerufen. Sie instanziiert zunächst ein Objekt des Typs `android.media.MediaPlayer` und weist es der In-stanzvariablen `player` zu. Anschließend wird ein `OnCompletionListener` registriert. Der Aufruf von dessen Methode `onCompletion()` signalisiert das Ende eines Abspielvorgangs. In diesem Fall werden alle Ressourcen, die durch das `MediaPlayer`-Objekt belegt werden, freigegeben.

Die Implementierung von `releasePlayer()` zeige ich Ihnen im Anschluss. Um eine Audiodatei wiederzugeben, müssen Sie zunächst die Quelle festlegen. Hierfür gibt es die Methode `setDataSource()`. `prepare()` bereitet das Abspielen vor. Mit `start()` beginnt die Wiedergabe.

```
private void playAudioFile(String filename) {
   player = new MediaPlayer();
   player.setOnCompletionListener(new OnCompletionListener() {
      public void onCompletion(MediaPlayer player) {
         releasePlayer();
         mode = MODE.WAITING;
         updateButtonText();
      }
   });
   try {
```

```
            player.setDataSource(filename);
            player.prepare();
            player.start();
            mode = MODE.PLAYING;
            updateButtonText();
        } catch (Throwable thr) {
            // IllegalArgumentException,
            // IllegalStateException
            // IOException
            Log.e(TAG, "could not play audio", thr);
        }
    }
```

Listing 14.4 Auszug aus RR.java

Das Freigeben von Ressourcen habe ich in die Methode `releasePlayer()` ausgelagert. Auf diese Weise kann sehr leicht geprüft werden, ob überhaupt eine `MediaPlayer`-Instanz aktiv ist:

```
private void releasePlayer() {
    if (player != null) {
        // Player-Ressourcen freigeben
        player.release();
        player = null;
    }
}
```

Listing 14.5 Auszug aus RR.java

Tipp

Sie können die hier beschriebene Vorgehensweise ohne Änderungen übernehmen, wenn Sie Musikdateien abspielen möchten. Alles, was Sie hierfür benötigen, ist in der Methode `playAudioFile()` zu finden.

Audiodateien aufzeichnen

Um eine Audiodatei aufzuzeichnen, müssen Sie als Erstes ein Objekt des Typs `android.media.MediaRecorder` instanziieren und anschließend durch Aufruf von dessen Methode `setAudioSource()` die Aufnahmequelle festlegen. Das Ausgabeformat setzen Sie mit `setOutputFormat()`. `setAudioEncoder()` bestimmt den zu verwendenden Codec.

Nachdem Sie den Recorder konfiguriert haben, legen Sie einen Dateinamen fest und instanziieren ein entsprechendes `File`-Objekt. Sie müssen die korrespondierende Datei noch anlegen, und zwar indem Sie die Methode `createNewFile()` aufrufen. `prepare()` bereitet die Aufnahme vor. Nach dem Aufruf von `start()` beginnt sie.

```
private File recordToFile() {
   recorder = new MediaRecorder();
   recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
   recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
   recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
   File f = new RRFile(getBaseDir(), Long.toString(System
         .currentTimeMillis()) + RRFile.EXT_3GP);

   try {
      f.createNewFile();
      recorder.setOutputFile(f.getAbsolutePath());
      recorder.prepare();
      recorder.start();
      mode = MODE.RECORDING;
      updateButtonText();
      return f;
   } catch (IOException e) {
      Log.e(TAG, "could not start recording", e);
   }
   return null;
}
```

Listing 14.6 Auszug aus RR.java

Um eine Aufnahme zu beenden, müssen Sie die Methode `stop()` Ihrer `MediaRecorder`-Instanz aufrufen. Anschließend sollten Sie die nicht mehr benötigten Ressourcen freigeben. Ich habe hierzu die Methode `releaseRecorder()` implementiert. Sie prüft analog zu `releasePlayer()`, ob überhaupt ein `MediaRecorder`-Objekt in Verwendung ist.

```
private void releaseRecorder() {
   if (recorder != null) {
      // Recorder-Ressourcen freigeben
      recorder.release();
      recorder = null;
   }
}
```

Listing 14.7 Auszug aus RR.java

`MediaRecorder` und `MediaPlayer` ermöglichen Ihnen den unkomplizierten Einstieg in die faszinierende Welt der Audioverarbeitung. Wie Sie auf diesen Grundlagen aufbauen, zeige ich Ihnen im folgenden Abschnitt. Er beschäftigt sich mit Audioeffekten.

14.2 Effekte

Google hat mit der Android-Version 2.3 (Gingerbread) zahlreiche Audioeffekte eingeführt. Wie Sie gleich sehen werden, lassen sich diese beliebig mischen und sowohl global als auch für einzelne Tonspuren zuschalten. Kopieren und importieren Sie zuerst das Projekt *AudioEffekteDemo* in Ihren Eclipse-Arbeitsbereich. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD des Buches.

14.2.1 Die Klasse AudioEffekteDemo

Nach dem Start des Programms spielen Sie einen kurzen Audioschnipsel ab, indem Sie die Schaltfläche START anklicken. Wie Sie in Abbildung 14.2 sehen, können Sie die Wiedergabe mit STOP anhalten. Klicken Sie abermals auf START, um den Abspielvorgang fortzusetzen.

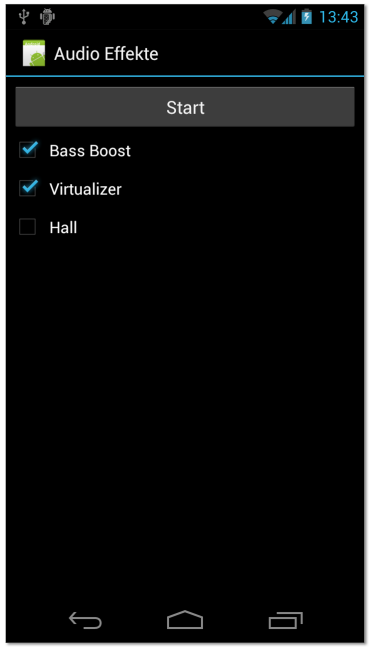


Abbildung 14.2 Die App »AudioEffekteDemo«

Die App demonstriert die drei Effekte BASS BOOST, Klangverbreiterung (VIRTUALIZER) und HALL. (Außer diesen drei Effekten kennt Android noch *Equalizer* und *Envi-*

ronmentalReverb.) Sie können sie in beliebiger Kombination zuschalten. Allerdings lässt sich mit dem Beispielsample nur HALL vernünftig demonstrieren.

Um eine eigene MP3-Datei abzuspielen, kopieren Sie diese in das Verzeichnis *res/raw*. Dort befindet sich bereits *guten_tag.mp3*. Da aus dem Dateinamen eine Konstante (zum Beispiel `R.raw.guten_tag`) erzeugt wird, sollte dieser keine Leerzeichen enthalten und idealerweise nur aus Kleinbuchstaben, Ziffern und dem Unterstrich bestehen. Damit die App Ihre Datei abspielt, müssen Sie in der Zeile `mediaPlayer = MediaPlayer.create(this, R.raw.guten_tag);` die Konstante entsprechend ersetzen.

Hinweis

Im Verzeichnis *res/raw* abgelegte Dateien werden ohne Änderungen oder Konvertierungen in die Installationsdatei (*.apk*) einer App übernommen. Sie können über Konstanten in `R.raw` auf sie zugreifen. Allerdings hat die Größe solcher Dateien direkten Einfluss auf die Größe der Installationsdatei. Sie sollten dies trotz stetig schneller werdender Datenverbindungen berücksichtigen.

Die Klasse `AudioEffekteDemo` implementiert die drei Methoden `onCreate()`, `onDestroy()` und `updateButtonText()`. Letztere setzt die Beschriftung des (einzigen) Buttons. Ob aktuell eine Audiodatei abgespielt wird, ist in der Instanzvariablen `playing` hinterlegt.

In `onCreate()` wird ein `MediaPlayer`-Objekt instanziiert und ein `OnCompletionListener` registriert. Das ist notwendig, um am Ende des Abspielvorgangs die Schaltfläche aktualisieren zu können. Die Freigabe des Players (`mediaPlayer.release()`) erfolgt aber erst, wenn die Activity zerstört wird. Dann ruft Android die Methode `onDestroy()` auf.

```
public class AudioEffekteDemo extends Activity {
   private static final String TAG = AudioEffekteDemo.class.getSimpleName();
   private MediaPlayer mediaPlayer;

   // Effekte
   private BassBoost bassBoost = null;
   private Virtualizer virtualizer = null;
   private PresetReverb reverb = null;

   private Button button;
   private boolean playing;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        // MediaPlayer instanziieren
        mediaPlayer = MediaPlayer.create(this, R.raw.guten_tag);
        mediaPlayer.setOnCompletionListener(new OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mp) {
                playing = false;
                updateButtonText();
            }
        });
        int sessionId = mediaPlayer.getAudioSessionId();
        // Effekte
        ...
        // Schaltfläche
        button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                if (playing) {
                    mediaPlayer.pause();
                } else {
                    mediaPlayer.start();
                }
                playing = !playing;
                updateButtonText();
            }
        });
        playing = false;
        updateButtonText();
    }

    private void updateButtonText() {
        button.setText(getString(playing ? R.string.stop : R.string.start));
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mediaPlayer.stop();
        if (bassBoost != null) {
            bassBoost.release();
        }
        if (virtualizer != null) {
            virtualizer.release();
```

```
        }
        if (reverb != null) {
           reverb.release();
        }
        mediaPlayer.release();
    }
}
```

Listing 14.8 Auszug aus AudioEffekteDemo.java

Vielleicht fragen Sie sich, warum der Variablen `sessionId` zwar das Ergebnis der Methode `getAudioSessionId()` zugewiesen, dieses aber nicht weiter verwendet wird. Die drei Punkte ... unterhalb des Kommentars `// Effekte` deuten an, dass an dieser Stelle noch Code fehlt. Die Audiosession-ID verknüpft Audioeffekte mit einem Audiostrom. Wie dies funktioniert, zeige ich Ihnen jetzt.

14.2.2 Bass Boost und Virtualizer

Bass Boost verstärkt niedrige Frequenzen eines Audiostroms. `android.media.audiofx.BassBoost` erbt von `android.media.audiofx.AudioEffect`, der Basisklasse aller Audioeffekte. Diese implementiert Standardverhalten, zum Beispiel das Ein- oder Ausschalten eines Effekts mit `setEnabled()` sowie das Freigeben von Ressourcen mit `release()`. Vor dem Start eines Abspielvorgangs durch `mediaPlayer.start()` müssen alle gewünschten Effekte mit `setEnabled(true)` aktiviert werden.

Nach dem Erzeugen eines `BassBoost`-Objekts lässt sich dessen Intensität mit `setStrength()` in einem Bereich zwischen 0 und 1.000 einstellen. Beachten Sie, dass der tatsächlich gesetzte vom übergebenen Wert abweichen kann, wenn die Implementierung keine entsprechend feine Abstufung unterstützt.

Um das zu prüfen, rufen Sie anschließend `getRoundedStrength()` auf. Über die Auskunftsmethode `getStrengthSupported()` können Sie herausfinden, ob die `BassBoost`-Implementierung das Einstellen der Intensität grundsätzlich zulässt. Ist dies nicht der Fall (der Rückgabewert ist `false`), kennt sie nur eine Stufe. Jeder Aufruf von `setStrength()` rundet dann auf den korrespondierenden Wert.

Die meisten Audioeffekte können entweder dem systemweiten, gemixten Audiostrom oder »nur« einer bestimmten `MediaPlayer`-Instanz hinzugefügt werden. Dies wird über den zweiten Parameter (`audioSession`) des Effekt-Konstruktors gesteuert. 0 kennzeichnet den globalen Mix. Damit das klappt, muss eine App in ihrer Manifestdatei die Berechtigung `android.permission.MODIFY_AUDIO_SETTINGS` anfordern.

Jeder andere Wert repräsentiert eine systemweit eindeutige AudioSession-ID. Sie können diese mit `mediaPlayer.getAudioSessionId()` ermitteln. Das folgende Quelltext-

fragment instanziiert ein `BassBoost`-Objekt und bindet es an einen `MediaPlayer`. Ob der Effekt zu hören ist, hängt vom Status des korrespondierenden Ankreuzfeldes ab.

```
// BassBoost instanziieren und an Audio Session binden
bassBoost = new BassBoost(0, sessionId);
Log.d(TAG, "getRoundedStrength(): " + bassBoost.getRoundedStrength());
if (bassBoost.getStrengthSupported()) {
   bassBoost.setStrength((short) 1000);
}

// Checkbox schaltet BassBoost aus und ein
final CheckBox cbBassBoost
         = (CheckBox) findViewById(R.id.cbBassBoost);
cbBassBoost.setOnCheckedChangeListener(
      new OnCheckedChangeListener() {
   public void onCheckedChanged(CompoundButton buttonView,
         boolean isChecked) {
      int result = bassBoost.setEnabled(isChecked);
      if (result != AudioEffect.SUCCESS) {
         Log.e(TAG, "Bass Boost: setEnabled("
               + isChecked + ") = "
               + result);
      }
   }
});
cbBassBoost.setChecked(false);
```

Listing 14.9 Auszug aus AudioEffekteDemo.java

Der Effekt VIRTUALIZER verändert die räumliche Wirkung eines Audiosignals. Wie er sich im Detail auswirkt, hängt von der Anzahl der Eingabe- sowie der Art und Anzahl der Ausgabekanäle ab. Eine in Stereo aufgenommene *.mp3*-Datei klingt über einen Kopfhörer breiter, wenn der Virtualizer aktiviert ist.

Das folgende Quelltextfragment instanziiert ein `Virtualizer`-Objekt und bindet es an einen `MediaPlayer`. Ob der Effekt zu hören ist, hängt vom Status des korrespondierenden Ankreuzfeldes ab.

```
// Virtualizer instanziieren und an Audio Session binden
virtualizer = new Virtualizer(0, sessionId);
virtualizer.setStrength((short) 1000);

// Checkbox schaltet Virtualizer aus und ein
final CheckBox cbVirtualizer = (CheckBox) findViewById(R.id.cbVirtualizer);
cbVirtualizer.setOnCheckedChangeListener(
```

```
      new OnCheckedChangeListener() {
   public void onCheckedChanged(CompoundButton buttonView,
         boolean isChecked) {
      int result = virtualizer.setEnabled(isChecked);
      if (result != AudioEffect.SUCCESS) {
         Log.e(TAG, "Virtualizer: setEnabled("
               + isChecked + ") = "
               + result);
      }
   }
});
cbVirtualizer.setChecked(false);
```

Listing 14.10 Auszug aus AudioEffekteDemo.java

14.2.3 Hall

Je nach räumlicher Umgebung wird Schall unterschiedlich oft und stark reflektiert. Große Konzertsäle haben selbstverständlich eine andere Akustik als beispielsweise Kathedralen oder kleine Zimmer. `android.media.audiofx.PresetReverb` implementiert einen solchen Effekt.

Anders als die beiden bereits vorgestellten Klassen `Virtualizer` und `BassBoost` wird `PresetReverb` immer an den globalen Mixer (Audio-Session 0) gebunden. Damit das Ausgabesignal einer `MediaPlayer`-Instanz eingespeist werden kann, ermitteln Sie zunächst mit `reverb.getId()` die Effekt-ID des `PresetReverb`-Objekts und übergeben diese an die `MediaPlayer`-Methode `attachAuxEffect()`. Anschließend setzen Sie noch den `setAuxEffectSendLevel()` und legen damit die Intensität fest.

Das folgende Quelltextfragment instanziiert ein `PresetReverb`-Objekt und bindet es an einen `MediaPlayer`. Ob der Effekt zu hören ist, hängt vom Status des korrespondierenden Ankreuzfeldes ab.

```
// Hall
reverb = new PresetReverb(0, 0);
int effectId = reverb.getId();
reverb.setPreset(PresetReverb.PRESET_PLATE);
mediaPlayer.attachAuxEffect(effectId);
mediaPlayer.setAuxEffectSendLevel(1f);

// Checkbox schaltet Hall aus und ein
final CheckBox cbReverb = (CheckBox) findViewById(R.id.cbReverb);
cbReverb.setOnCheckedChangeListener(
      new OnCheckedChangeListener() {
```

```
    public void onCheckedChanged(CompoundButton buttonView,
          boolean isChecked) {
       int result = reverb.setEnabled(isChecked);
       if (result != AudioEffect.SUCCESS) {
          Log.e(TAG, "PresetReverb: setEnabled("
                + isChecked + ") = "
                + result);
       }
    }
});
cbReverb.setChecked(false);
```

Listing 14.11 Auszug aus AudioEffekteDemo.java

Hinweis

Unglücklicherweise scheinen einige Versionen des Emulators Probleme mit Audioeffekten zu haben. Dies äußert sich in *LogCat*-Meldungen wie `AudioFlinger could not create effect, status: -22`. In diesem Fall bleibt Ihnen leider nur, die App auf echter Hardware auszuprobieren.

Soundeffekte verleihen Musik-Apps nicht nur den nötigen Pepp – sie sind schlicht ein Muss. Übrigens stellt Android einige weitere interessante Klassen im Bereich Audioverarbeitung zur Verfügung. Mit `android.media.audiofx.Visualizer` können Sie beispielsweise am Ausgabestrom lauschen, um diesen zu visualisieren. Denken Sie an Zeigerinstrumente früherer HiFi-Tage oder opulente Farbspiele. Experimentieren Sie mit den Programmierschnittstellen, und bauen Sie Live Wallpapers oder Widgets.

14.3 Sprachsynthese

Schon seit Android 1.6 (Donut) enthält die Plattform die Sprachsynthesekomponente *Pico*. Diese kann Texte in Deutsch, Englisch, Italienisch, Französisch und Spanisch vorlesen. In diesem Abschnitt zeige ich Ihnen, wie Sie Ihre Apps um die Fähigkeit erweitern, mit dem Anwender zu sprechen. Wie praktisch das ist, wissen wir spätestens durch die Benutzung von Navigationshilfen. Aber die Einsatzmöglichkeiten sind weitaus vielschichtiger. Lassen Sie sich doch Ihre E-Mails vorlesen. Oder programmieren Sie eine App, die Ihnen auf Knopfdruck die aktuelle Uhrzeit ansagt.

14.3.1 Nutzung der Sprachsynthesekomponente vorbereiten

Welche Schritte notwendig sind, um geschriebene in gesprochene Sprache umzuwandeln, möchte ich Ihnen anhand der App *TextToSpeechDemo* zeigen. Wie üblich finden Sie das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich. Bevor ich Sie mit der Implementierung vertraut mache, sollten Sie die App kurz starten. Die Benutzeroberfläche sehen Sie in Abbildung 14.3.

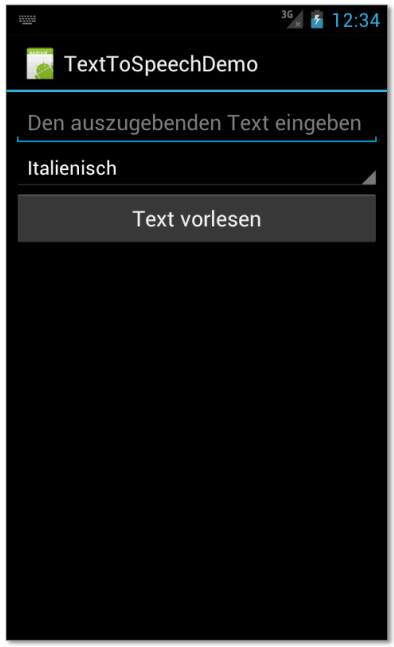


Abbildung 14.3 Benutzeroberfläche der App »TextToSpeechDemo«

Tippen Sie als Erstes einen beliebigen Text in das Eingabefeld, und wählen Sie dann die gewünschte Sprache. Abbildung 14.4 zeigt, welche Ihnen zur Verfügung stehen. TEXT VORLESEN startet den Synthesevorgang und anschließend die Wiedergabe. Während der Text vorgelesen wird, ist die Schaltfläche nicht anwählbar.

Die Klasse `TextToSpeechDemo` leitet von `android.app.Activity` ab und implementiert die üblichen Methoden `onCreate()` und `onDestroy()`. Wenn Sie einen Blick auf das folgende Quelltextfragment werfen, fällt Ihnen aber sicher auf, dass `onCreate()` nicht wie üblich die Benutzeroberfläche initialisiert und anzeigt. Stattdessen wird »nur« die Variable `tts` mit `null` belegt und mit `start-ActivityForResult()`eine neue Activity aufgerufen.

`tts` zeigt auf ein Objekt des Typs `android.speech.tts.TextToSpeech`. Diese Klasse bildet die Zugriffsschicht auf die Sprachsynthesekomponente.

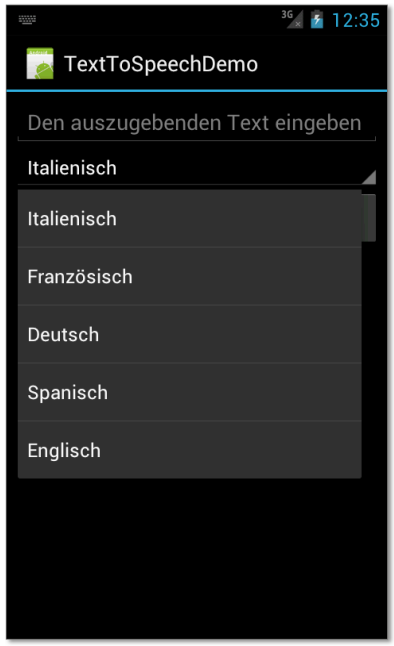


Abbildung 14.4 Sprache auswählen, in der vorgelesen werden soll

Die Methode `onDestroy()` gibt alle Ressourcen frei, die von der Syntheseeinheit reserviert wurden. Hierzu ruft sie deren Methode `shutdown()` auf.

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Die Sprachsynthesekomponente wurde
   // noch nicht initialisiert
   tts = null;
   // prüfen, ob Sprachpakete vorhanden sind
   Intent intent = new Intent();
   intent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
   startActivityForResult(intent, RQ_CHECK_TTS_DATA);
}

@Override
protected void onDestroy() {
   super.onDestroy();
   // ggf. Ressourcen freigeben
   if (tts != null) {
```

```
      tts.shutdown();
   }
}
```

Listing 14.12 Auszug aus TextToSpeechDemo.java

Installation prüfen

Die eigentliche Sprachsynthesesoftware ist zwar integraler Bestandteil der Android-Plattform, allerdings stand und steht es Geräteherstellern frei, aus Speicherplatzgründen die Sprachdaten nicht vorzuinstallieren. Deshalb ist es bewährte Praxis, einen Intent mit der Action `TextToSpeech.Engine.ACTION_CHECK_TTS_DATA` zu versenden, um deren Verfügbarkeit zu prüfen.

Meldet das System, dass alles in Ordnung ist, kann das Text-to-Speech-Modul initialisiert werden. Andernfalls müssen Sie mit einem weiteren Intent den Download der fehlenden Daten initiieren. Wie das funktioniert, zeigt das folgende Quelltextfragment.

Die Methode `onActivityResult()` erhält als zweiten Parameter einen `requestCode`. Dieser wird neben dem Intent an `startActivityForResult()` übergeben, kann von Ihnen also frei gewählt werden. Der `resultCode` gibt an, ob der Aufruf der Activity erfolgreich war. `TextToSpeech.Engine.CHECK_VOICE_DATA_PASS` bedeutet, dass alle benötigten Bestandteile vorhanden sind.

Ist das der Fall, kann die Synthesesoftware initialisiert werden. Dies geschieht mit dem Ausdruck `tts = new TextToSpeech(this, this);`. Andernfalls sollten Sie die Installation der fehlenden Komponenten anstoßen. Mein Beispiel startet hierzu eine neue Activity und beendet die aktuelle mit `finish()`. Alternativ können Sie abermals `startActivityForResult()` aufrufen.

```
protected void onActivityResult(int requestCode,
      int resultCode, Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   // Sind Sprachpakete vorhanden?
   if (requestCode == RQ_CHECK_TTS_DATA) {
      if (resultCode == TextToSpeech.Engine.CHECK_VOICE_DATA_PASS) {
         // Initialisierung der Sprachkomponente starten
         tts = new TextToSpeech(this, this);
      } else {
         // Installation der Sprachpakete vorbereiten
         Intent installIntent = new Intent();
         installIntent
               .setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
         startActivity(installIntent);
```

```
            // Activity beenden
            finish();
        }
    }
}
```

Listing 14.13 Auszug aus TextToSpeechDemo.java

Das Interface android.speech.tts.TextToSpeech.OnInitListener

Das Sprachausgabemodul darf erst nach einer erfolgreichen Initialisierung verwendet werden. Aus diesem Grund wird dem Konstruktor der Klasse `TextToSpeech` als zweites Argument ein Objekt des Typs `android.speech.tts.TextToSpeech.OnInitListener` übergeben. Da die Activity `TextToSpeechDemo` dieses Interface implementiert, finden Sie an dieser Stelle ein `this`.

Die im Folgenden dargestellte Implementierung der Methode `onInit()` erledigt einige Aufgaben, die üblicherweise in `onCreate()` ausgeführt werden, beispielsweise das Anzeigen der Benutzeroberfläche mit `setContentView()`. Dies ist notwendig, weil unter anderem die Klappliste für die Sprachauswahl erst nach einer erfolgreichen Initialisierung der Synthesekomponente gefüllt werden kann.

```
@Override
public void onInit(int status) {
    if (status != TextToSpeech.SUCCESS) {
        // die Initialisierung war nicht erfolgreich
        finish();
    }
    // Activity initialisieren
    setContentView(R.layout.main);
    input = (EditText) findViewById(R.id.input);
    spinner = (Spinner) findViewById(R.id.locale);
    button = (Button) findViewById(R.id.button);
    button.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            String text = input.getText().toString();
            String key = (String) spinner.getSelectedItem();
            Locale loc = supportedLanguages.get(key);
            if (loc != null) {
                button.setEnabled(false);
                tts.setLanguage(loc);
                HashMap<String, String> hashmap
                    = new HashMap<String, String>();
```

```
            last_utterance_id =
               Long.toString(System.currentTimeMillis());
            hashmap.put(
               TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,
               last_utterance_id);
            tts.speak(text, TextToSpeech.QUEUE_FLUSH, hashmap);
            // in Datei schreiben
            String filename = new File(
                 Environment.getExternalStorageDirectory(),
                 last_utterance_id + ".wav").getAbsolutePath();
            tts.synthesizeToFile(text, hashmap, filename);
         }
      }
});
tts.setOnUtteranceProgressListener(new UtteranceProgressListener() {

   @Override
   public void onStart(String utteranceId) {
      Log.d(TAG, "onStart(): " + utteranceId);
   }

   @Override
   public void onDone(final String utteranceId) {
      final Handler h = new Handler(Looper.getMainLooper());
      h.post(new Runnable() {
         @Override
         public void run() {
            if (utteranceId.equals(last_utterance_id)) {
               button.setEnabled(true);
            }
         }
      });
   }

   @Override
   public void onError(String utteranceId) {
      Log.d(TAG, "onError(): " + utteranceId);
   }
});
// Liste der Sprachen ermitteln
String[] languages = Locale.getISOLanguages();
for (String lang : languages) {
```

```
            Locale loc = new Locale(lang);
            switch (tts.isLanguageAvailable(loc)) {
            case TextToSpeech.LANG_MISSING_DATA:
            case TextToSpeech.LANG_NOT_SUPPORTED:
               break;
            default:
               String key = loc.getDisplayLanguage();
               if (!supportedLanguages.containsKey(key)) {
                  supportedLanguages.put(key, loc);
               }
               break;
            }
         }
         ArrayAdapter<Object> adapter = new ArrayAdapter<Object>(this,
               android.R.layout.simple_spinner_item,
               supportedLanguages.keySet().toArray());
         adapter.setDropDownViewResource(
               android.R.layout.simple_spinner_dropdown_item);
         spinner.setAdapter(adapter);
      }
```

Listing 14.14 Auszug aus TextToSpeechDemo.java

Da ich zu Beginn dieses Abschnitts die fünf Sprachen aufgezählt habe, die Pico derzeit unterstützt, fragen Sie sich vielleicht, warum ich diese nicht einfach »fest verdrahte«. Ganz einfach: Sollten irgendwann zusätzliche Sprachen hinzukommen, erkennt mein Programm diese automatisch.

Tipp

Treffen Sie niemals Annahmen über die Beschaffenheit eines Systems, wenn es stattdessen Auskunftsfunktionen gibt.

Beispielsweise können Sie mit `isLanguageAvailable()` prüfen, ob eine bestimmte Sprache der Synthesekomponente zur Verfügung steht.

14.3.2 Texte vorlesen

Um einen Text vorlesen zu lassen, rufen Sie die Methode `speak()` eines `TextToSpeech`-Objekts auf. Ihr wird unter anderem eine `HashMap` übergeben, die wichtige Daten für die Ausgabe enthält. Die Utterance-ID ist eine eindeutige Kennung, mit der der vorzulesende Text identifiziert wird.

```
String text = input.getText().toString();
String key = (String) spinner.getSelectedItem();
Locale loc = supportedLanguages.get(key);
if (loc != null) {
   button.setEnabled(false);
   tts.setLanguage(loc);
   HashMap<String, String> hashmap
         = new HashMap<String, String>();
   last_utterance_id = Long.toString(System.currentTimeMillis());
   hashmap.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,
         last_utterance_id);
   tts.speak(text, TextToSpeech.QUEUE_FLUSH, hashmap);
}
```

Listing 14.15 Auszug aus TextToSpeechDemo.java

Ist Ihnen in der Implementierung der Methode `onInit()` der Aufruf der Methode `setOnUtteranceProgressListener()` aufgefallen? Die abstrakte Klasse `UtteranceProgressListener` enthält die Methoden `onStart()`, `onDone()` und `onError()`, die im Verlauf einer Sprachausgabe angesprungen werden können. Beispielsweise signalisiert der Aufruf von `onDone()`, dass eine Textausgabe vollständig durchgeführt wurde. Welche das war, können Sie anhand der Utterance-ID erkennen. Die Implementierung in der Klasse `TextToSpeechDemo` macht die Schaltfläche TEXT VORLESEN wieder anwählbar.


```
@Override
public void onDone(final String utteranceId) {

   final Handler h = new Handler(Looper.getMainLooper());
   h.post(new Runnable() {
      @Override
      public void run() {
         if (utteranceId.equals(last_utterance_id)) {
            button.setEnabled(true);
         }
      }
   });
}
```

Listing 14.16 Auszug aus TextToSpeechDemo.java

Wenn Sie in `onDone()` den Status von Bedienelementen verändern möchten, sollten Sie sicherstellen, dass Ihre Anweisungen auf dem UI-Thread ausgeführt werden. Instanziieren Sie hierzu mit

```
new Handler(Looper.getMainLooper())
```

ein Objekt des Typs `android.os.Handler`, und rufen Sie dessen `post()`-Methode auf. Ausführliche Informationen hierzu finden Sie in Kapitel 6, »Multitasking«.

14.3.3 Sprachausgaben speichern

Sie können `TextToSpeech`-Objekte nicht nur für die unmittelbare Ausgabe von Sprache verwenden. Die Klasse enthält die Methode `synthesizeToFile()`, die das akustische Signal als *.wav*-Datei auf dem externen Speichermedium ablegt. Um das auszuprobieren, fügen Sie das folgende Quelltextfragment nach dem Aufruf von `tts.speak()` ein:

```
// in Datei schreiben
String filename = new File(Environment.getExternalStorageDirectory(),
      last_utterance_id + ".wav").getAbsolutePath();
tts.synthesizeToFile(text, hashmap, filename);
```

Listing 14.17 Sprachsignal in Datei schreiben

Nach dem erneuten Start von *TextToSpeechDemo* geben Sie einen beliebigen Text ein, und wählen Sie die gewünschte Sprache. Klicken Sie anschließend auf TEXT VORLESEN. Sobald Android die Sprachausgabe beendet, erscheint die Datei, wie Sie in Abbildung 14.5 sehen können, in der Sicht FILE EXPLORER.

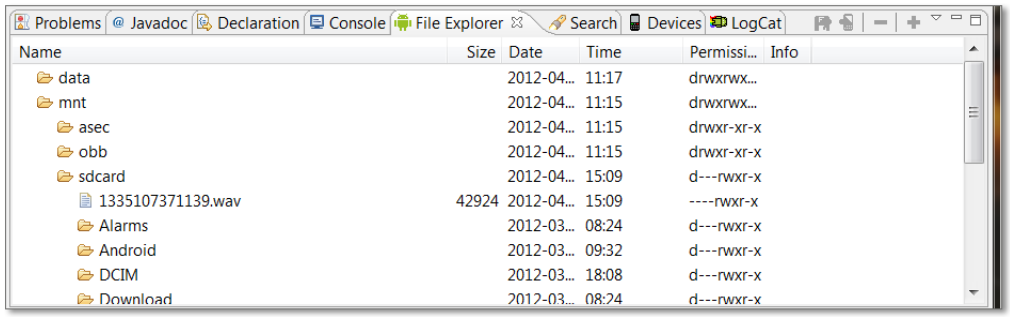


Abbildung 14.5 Die mit »synthesizeToFile()« erzeugte Datei

Im folgenden Abschnitt möchte ich Sie mit einigen weiteren, äußerst spannenden Audiofunktionen von Android vertraut machen.

14.4 Weitere Audiofunktionen

Die Plattform kann nicht nur Sprache synthetisieren, sondern auch erkennen. Ihre App könnte also ausschließlich in gesprochener Sprache mit dem Anwender kommunizieren. Nutzer von entsprechend ausgerüsteten Navigationssystemen haben diese Form der Bedienung schnell schätzen gelernt. Aber auch in vielen anderen Bereichen ist Spracherkennung äußerst praktisch. Denken Sie an das automatische Wählen nach Nennung eines Kontakts oder die Steuerung des Telefons durch einfache mündliche Anweisungen.

Im Folgenden zeige ich Ihnen daher, wie Sie die in Android integrierte Spracherkennungsfunktion in Ihren Apps nutzen. Außerdem lernen Sie eine sehr interessante Funktion der kabelgebundenen Headsets kennen, die ja praktisch allen Android-Geräten beiliegen. Diese senden nämlich Tastendrücke, auf die Sie in Ihren Programmen reagieren können.

14.4.1 Spracherkennung

Die App *SpracherkennungsDemo* zeigt, wie einfach sich diese Technologie in eigenen Programmen einsetzen lässt. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Kopieren und importieren Sie es wie üblich in Ihren Eclipse-Arbeitsbereich. Um die Anwendung ausprobieren zu können, benötigen Sie allerdings ein entsprechend ausgerüstetes Android-Smartphone. Die zum Zeitpunkt der Drucklegung vorhandenen virtuellen Geräte stellen die benötigte Funktionalität nämlich nicht zur Verfügung, so dass ein Test im Emulator leider nicht möglich ist.

Nach dem Start begrüßt *SpracherkennungsDemo* den Benutzer mit einem bis auf die Schaltfläche SPRACHERKENNUNG STARTEN leeren Bildschirm. Falls die benötigte Komponente nicht zur Verfügung steht, lautet die Beschriftung SPRACHERKENNUNG NICHT VERFÜGBAR. Der Button kann in diesem Fall nicht angeklickt werden. Ein Klick auf die Schaltfläche startet die Spracheingabe, deren Benutzeroberfläche in Abbildung 14.6 dargestellt ist.

Im Anschluss daran wird der Audiostrom an einen Google-Server geschickt und dort ausgewertet. Für die Spracherkennung ist also eine aktive Netzwerkverbindung erforderlich. Nach der Rückübertragung auf das Gerät zeigt die App analog Abbildung 14.7 die erkannten Worte an.

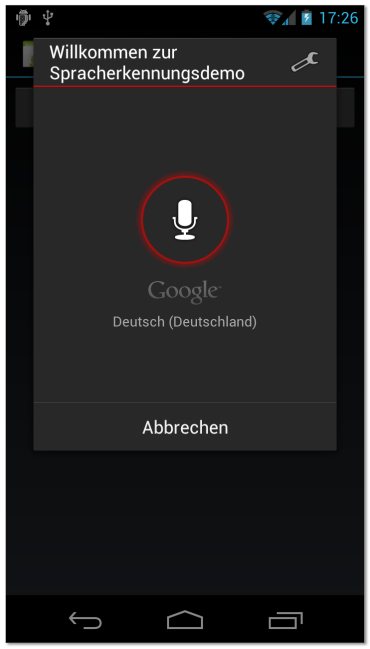


Abbildung 14.6 Der Aufnahmevorgang

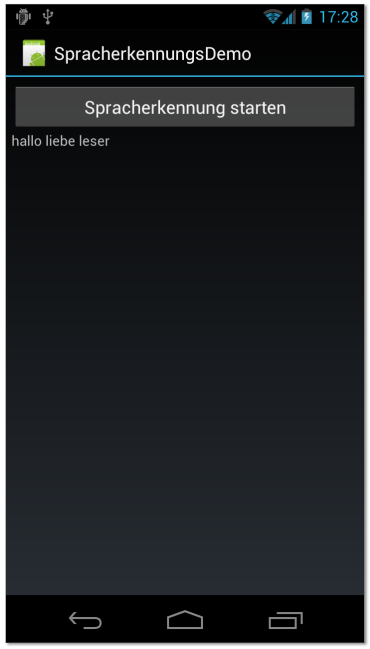


Abbildung 14.7 Die App zeigt den erkannten Text an.

Verfügbarkeit prüfen

Die Klasse `SpracherkennungsDemo` leitet von `android.app.Activity` ab und überschreibt deren beide Methoden `onCreate()` und `onActivityResult()`. Nach dem Setzen der Benutzeroberfläche wird eine Liste von Activities ermittelt, die das Intent `RecognizerIntent.ACTION_RECOGNIZE_SPEECH` verarbeiten können. Ist diese leer, steht die Spracherkennung nicht zur Verfügung.

Hinweis

Die Google-App *Sprachsuche* enthält eine Activity, die auf das Intent `RecognizerIntent.ACTION_RECOGNIZE_SPEECH` reagiert. Grundsätzlich kann aber jedes andere Programm ebenfalls darauf reagieren. Nutzt es eine andere Technologie, ist prinzipiell auch eine Offline-Erkennung möglich. Es bleibt abzuwarten, ob sich Alternativen zu Googles Sprachsuche etablieren.

Die Klasse `android.speech.RecognizerIntent` enthält zahlreiche weitere Konstanten, auf die ich gleich noch ausführlicher eingehen werde.


```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Benutzeroberfläche anzeigen
   setContentView(R.layout.main);
   Button button = (Button) findViewById(R.id.button);
   button.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
         startVoiceRecognitionActivity();
      }
   });

   textview = (TextView) findViewById(R.id.textview);
   // Verfügbarkeit der Spracherkennung prüfen
   PackageManager pm = getPackageManager();
   List<ResolveInfo> activities = pm.queryIntentActivities(
         new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
   if (activities.size() == 0) {
      button.setEnabled(false);
      button.setText(getString(R.string.not_present));
   }
}
```

Listing 14.18 Auszug aus SpracherkennungsDemo.java

Die Spracherkennung wird in der privaten Methode `startVoiceRecognitionActivity()` gestartet. Diese wird nach dem Anklicken der Schaltfläche SPRACHERKENNUNG STARTEN aufgerufen.

Spracherkennung starten

Wie Sie aus dem vorherigen Abschnitt wissen, wird die Spracherkennung durch eine (aus der Sicht Ihrer App beliebige) Activity ausgeführt, die das Intent `RecognizerIntent.ACTION_RECOGNIZE_SPEECH` verarbeiten kann. Da die Activity ein Ergebnis (nämlich die hoffentlich erkannten Wörter) an Sie übermitteln muss, starten Sie sie mit `startActivityForResult()`. Mit `putExtra()` können Sie deren Funktionsweise beeinflussen. Beispielsweise legen Sie mit `RecognizerIntent.EXTRA_LANGUAGE` die Sprache fest.

Mit `RecognizerIntent.EXTRA_MAX_RESULTS` steuern Sie die maximale Anzahl an Ergebnissen. Besonders wichtig ist `RecognizerIntent.EXTRA_LANGUAGE_MODEL`. Hiermit charakterisieren Sie die bevorstehende Eingabe. `LANGUAGE_MODEL_FREE_FORM` bedeutet, dass der Anwender eher ganze Sätze formulieren wird. Denken Sie an einen Texteditor, mit dem Sie E-Mails oder Kurznachrichten erfassen. `LANGUAGE_MODEL_WEB_SEARCH` hingegen verwenden Sie bei Suchanfragen oder grammatikalisch unvollständigen Verb-Substantiv-Phrasen.

```
private void startVoiceRecognitionActivity() {
   Intent intent =
         new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
   intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
         RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
   intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
         getString(R.string.prompt));
   intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "de-DE");
   intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 1);
   startActivityForResult(intent, RQ_VOICE_RECOGNITION);
}
```

Listing 14.19 Auszug aus SpracherkennungsDemo.java

Wenn die Spracherkennungs-Activity ihre Arbeit beendet hat, ruft Android die Methode `onActivityResult()` auf. Wie üblich müssen Sie zunächst die beiden Werte `requestCode` und `resultCode` prüfen. `getStringArrayListExtra()` ermittelt dann eine Liste mit Ergebnissen.

```
@Override
protected void onActivityResult(int requestCode,
      int resultCode, Intent data) {
```

```
    if (requestCode == RQ_VOICE_RECOGNITION
          && resultCode == RESULT_OK) {
       ArrayList<String> matches =
          data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
       if (matches.size() > 0) {
          textview.setText(matches.get(0));
       }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

Listing 14.20 Auszug aus SpracherkennungsDemo.java

Die Kombination von Sprachsynthese und Spracherkennung kann die Bedienung von mobilen Geräten enorm vereinfachen. Haben Sie Ideen? Ich freue mich schon auf Ihre Umsetzungen. Weitere Informationen finden Sie in Googles Entwickler-Dokumentation unter *Speech Input*.[1] Sie haben in diesem Kapitel schon zahlreiche Facetten der Audioverarbeitung unter Android kennengelernt. Mit einem weiteren spannenden Beispiel möchte ich diesen Bereich beschließen.

14.4.2 Tastendrücke von Headsets verarbeiten

Die meisten Hersteller liefern ihre Geräte mit einem kabelgebundenen Headset aus. Üblicherweise wird dieses zur Steuerung des Musikabspielprogramms verwendet. Wie Sie gleich sehen werden, nutzt Android hierfür ein anwendungsübergreifendes Protokoll, so dass sich auch Ihre eigene Audio-App damit bedienen lässt. Kopieren und importieren Sie das Projekt *AudioManagerDemo* in Ihren Eclipse-Arbeitsbereich. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD.

Wird eine Taste am Headset gedrückt, versendet die Plattform ein Broadcast-Event mit der Action `Intent.ACTION_MEDIA_BUTTON`. Dies lässt sich mit dem Emulator derzeit nicht simulieren, so dass Sie für den Test leider echte Hardware benötigen.

Nach dem Start der App sehen Sie einen leeren Bildschirm. Ausgaben finden in diesem Beispiel der Einfachheit halber ausschließlich in der in Abbildung 14.8 gezeigten Sicht LogCat statt. Sie können dies ausprobieren, indem Sie die Taste bzw. die Tasten Ihrer Fernbedienung lange oder kurz gedrückt halten.

Die Activity `AudioManagerDemo` überschreibt die drei Methoden `onPause()`, `onResume()` und `onCreate()`. Letztere ermittelt die Referenz auf ein Objekt des Typs `android.media.AudioManager`.

1 *http://developer.android.com/resources/articles/speech-input.html*

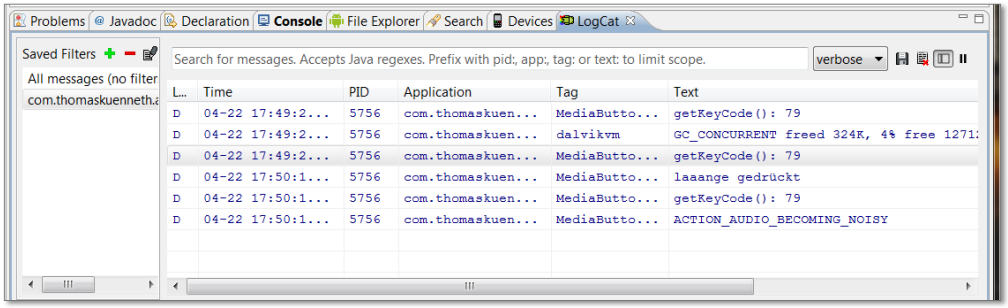


Abbildung 14.8 Ausgaben der App »AudioManagerDemo«

Diese Klasse bietet neben zahlreichen Methoden zur Konfiguration des Audio-Routings (die Sie üblicherweise nicht nutzen werden) unter anderem die Möglichkeit, das Telefon stumm zu schalten (`setRingerMode()`). Außerdem können Sie beispielsweise mit `isMusicActive()` abfragen, ob eine App Audio abspielt.

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Referenz auf AudioManager holen
   audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
   // ComponentName erzeugen
   eventReceiver = new ComponentName(this,
         MediaButtonEventReceiver.class);
}
```

Listing 14.21 Auszug aus AudioManagerDemo.java

Um über Tastendrücke am kabelgebundenen Headset informiert zu werden, muss die App mit `registerMediaButtonEventReceiver()` einen Broadcast Receiver registrieren. Sollen entsprechende Meldungen nicht mehr erfolgen, rufen Sie einfach die Methode `unregisterMediaButtonEventReceiver()` auf.

Vielleicht fragen Sie sich nun, wann hierfür ein guter Zeitpunkt ist. Solange die Benutzeroberfläche einer App zu sehen ist, erwartet der Anwender, dass er mit dem Headset dieses Programm bedient. Deshalb rate ich Ihnen, den Receiver in `onResume()` zu registrieren und in `onPause()`wieder zu entfernen.

```
@Override
protected void onResume() {
  super.onResume();
  audioManager.registerMediaButtonEventReceiver(eventReceiver);
}
```

```
@Override
protected void onPause() {
  super.onPause();
  audioManager.unregisterMediaButtonEventReceiver(eventReceiver);
}
```

Listing 14.22 Auszug aus AudioManagerDemo.java

Natürlich kann es auch sinnvoll sein, dass Hintergrundprozesse die Fernbedienung für sich beanspruchen. Denken Sie in diesem Fall bei der Konzeption aber an die Erwartungshaltung des Anwenders. Die Plattform unterstützt Sie dabei, indem einer App so lange Tastendrücke zugestellt werden, bis sich ein anderes Programm registriert. Dann erhält dieses die Broadcast-Events. Ruft eine App `unregisterMediaButtonEventReceiver()` auf, kommt wieder diejenige zum Zuge, die zuletzt `registerMediaButtonEventReceiver()` aufgerufen hatte.

Wie Sie aus Kapitel 4, »Activities und Broadcast Receiver«, wissen, müssen Broadcast Receiver in der Manifestdatei eingetragen werden. Sie teilen der Plattform mit dem Kindelement `<intent-filter />` mit, auf welche Intents der Receiver reagieren möchte.


```
<receiver android:name=".MediaButtonEventReceiver">
<intent-filter>
   <action android:name="android.intent.action.MEDIA_BUTTON" />
</intent-filter>
</receiver>
```

Listing 14.23 Auszug aus der Manifestdatei von AudioManagerDemo

Die Klasse `MediaButtonEventReceiver` leitet von `android.content.BroadcastReceiver` ab und implementiert die Methode `onReceive()`. Diese prüft zunächst, ob die Action des übermittelten Intents die gewünschte `Intent.ACTION_MEDIA_BUTTON` ist. Trifft dies zu, enthält das Intent zusätzlich ein Objekt des Typs `android.view.KeyEvent`.

```
public class MediaButtonEventReceiver extends BroadcastReceiver {

   private static final String TAG =
      MediaButtonEventReceiver.class.getSimpleName();

   @Override
   public void onReceive(Context context, Intent intent) {
      if (intent != null) {
         String action = intent.getAction();
         if (Intent.ACTION_MEDIA_BUTTON.equals(action)) {
            KeyEvent keyEvent = (KeyEvent)
```

```
                intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
            if (KeyEvent.ACTION_UP == keyEvent.getAction()) {
                Log.d(TAG, "getKeyCode(): " + keyEvent.getKeyCode());
            }
            if (keyEvent.isLongPress()) {
                Log.d(TAG, "laaange gedrückt");
            }
        } else if (AudioManager.ACTION_AUDIO_BECOMING_NOISY
                    .equals(action)) {
            Log.d(TAG, "ACTION_AUDIO_BECOMING_NOISY");
        }
    }
  }
}
```

Listing 14.24 Auszug aus MediaButtonEventReceiver.java

Mit `getKeyCode()` können Sie den Code der auslösenden Taste erfragen. Selbst wenn ein Headset nur eine Taste hat, können Sie durch die Verwendung von `isLongPress()` zwei Funktionen auslösen.

Wenn Sie den Stecker des Headsets aus der Buchse ziehen (oder beispielsweise die Verbindung zu einem Bluetooth-Kopfhörer trennen), wird in der Sicht *LogCat* die Meldung `ACTION_AUDIO_BECOMING_NOISY` ausgegeben. Android signalisiert interessierten Programmen auf diese Weise, dass der Audiostrom in Kürze über den Lautsprecher ausgegeben wird. In diesem Fall ist es möglicherweise eine gute Idee, die Wiedergabe zu pausieren. Um entsprechende Broadcast Events zu empfangen, tragen Sie in der Manifestdatei den gewünschten Receiver ein.

```
<receiver android:name=".MediaButtonEventReceiver">
...
<intent-filter>
   <action android:name="android.media.AUDIO_BECOMING_NOISY" />
</intent-filter>
</receiver>
```

Die Abfrage erfolgt in der Methode `onReceive()` der Klasse `MediaButtonEventReceiver`. Sie ist im vorangehenden Listing zu sehen. Kabelgebundene Headsets lassen sich also sehr einfach in eigene Apps integrieren. Versuchen Sie doch, den *Rasenden Reporter* entsprechend zu erweitern. Oder starten Sie nach einem langen Druck auf die Taste eine Spracherkennungssitzung. Android bietet unzählige Möglichkeiten. Nutzen Sie Ihre Kreativität, und kombinieren Sie die in diesem Kapitel angesprochenen Aspekte zu etwas ganz Neuem.

Kapitel 15
Fotos und Video

Die Fähigkeit, Fotos und Videos aufzunehmen, ist für moderne Smartphones und Tablets selbstverständlich. Wie Sie die Kamerahardware in Ihren eigenen Apps nutzen, zeige ich Ihnen in diesem Kapitel.

Die Zeiten, in denen die Kameras in Mobiltelefonen gerade einmal zu Spaßfotos taugten, sind glücklicherweise längst vorbei. Aktuelle Android-Geräte machen durchweg gute Bilder und Videos. Dabei sind die Möglichkeiten, die sich aus dem Einsatz der Kamerahardware ergeben, praktisch grenzenlos. Denken Sie an die Überlagerung von Live-Daten mit computergenerierten Grafiken oder Informationen aus dem Netz (*Augmented Reality*) oder die Erkennung von Sehenswürdigkeiten mit Google Goggles.[1]

15.1 Vorhandene Activities nutzen

Wie Sie aus Kapitel 4, »Activities und Broadcast Receiver«, wissen, lassen sich mit Intents Komponenten unterschiedlicher Apps kombinieren. Die Nutzung vorhandener Bausteine hat für Sie als Entwickler den Vorteil, das Rad nicht neu erfinden zu müssen. Und der Anwender findet sich schneller zurecht, weil er die Bedienung einer wiederverwendeten Komponente bereits kennt.

15.1.1 Kamera-Activity starten

Wie leicht Sie die in Android eingebaute Kamera-App nutzen können, möchte ich Ihnen anhand des Programms *KameraDemo1* zeigen. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich. Nach dem Start sehen Sie die in Abbildung 15.1 gezeigte Benutzeroberfläche.

1 *www.google.com/mobile/goggles/*

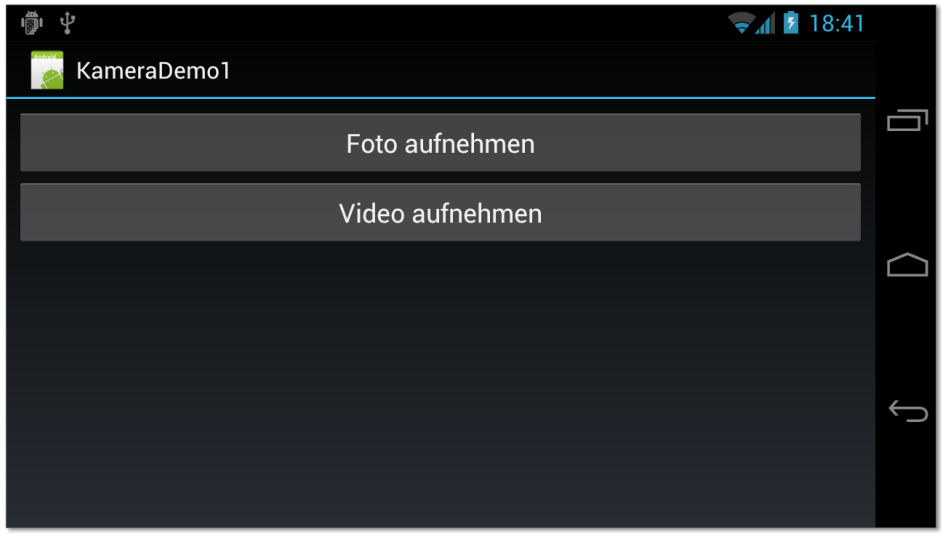


Abbildung 15.1 Die Benutzeroberfläche von »KameraDemo1«

Klicken Sie auf FOTO AUFNEHMEN, um die Kamera-App im Still-Image-Modus zu betreiben. Diesen sehen Sie in Abbildung 15.2.

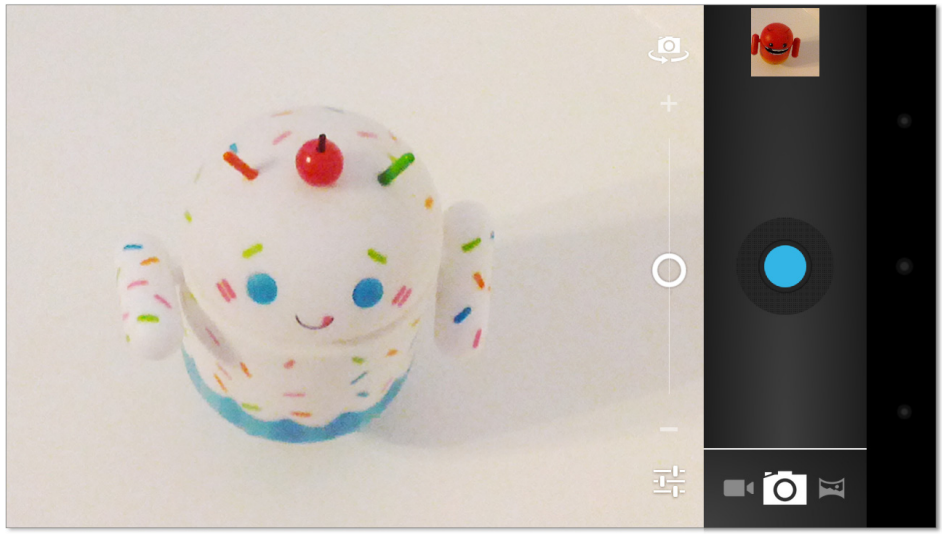


Abbildung 15.2 Die Kamera-App im Still-Image-Modus

Mit VIDEO AUFNEHMEN drehen Sie Filme. Zwischen diesem Videomodus, der in Abbildung 15.3 dargestellt ist, und dem Fotomodus kann der Anwender jederzeit umschalten. Beachten Sie in diesem Zusammenhang, dass die Benutzeroberfläche auf unterschiedlichen Geräten zum Teil stark variiert.

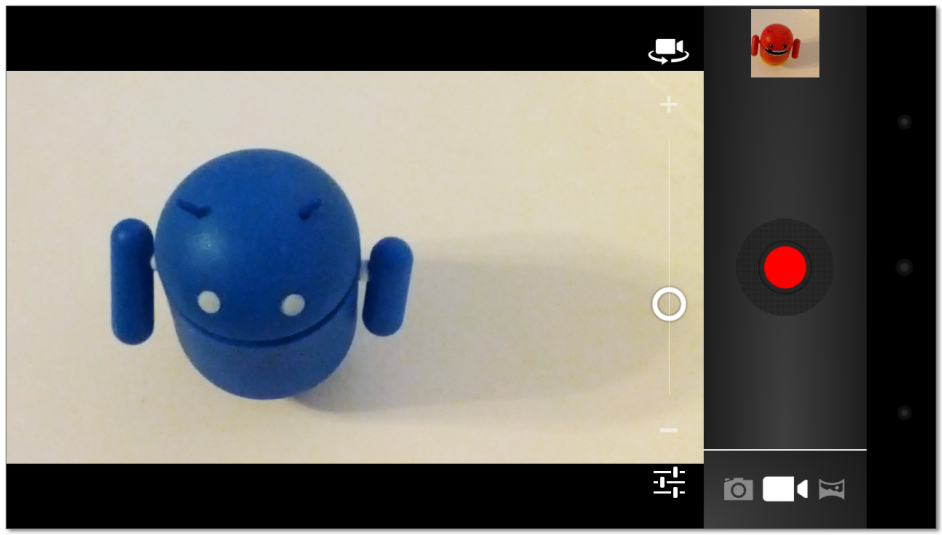


Abbildung 15.3 Die Kamera-App im Videomodus

Die Klasse `KameraDemo1` leitet von `android.app.Activity` ab und überschreibt nur `onCreate()`. In dieser Methode wird die Benutzeroberfläche gesetzt. Das Starten der Kamera-App findet in zwei Implementierungen von `onClick()` statt. Wie Sie bereits wissen, werden Instanzen des Typs `android.view.View.OnClickListener` verwendet, um auf das Anklicken von Schaltflächen zu reagieren.

Die Klasse `android.provider.MediaStore` gestattet den Zugriff auf die Mediendatenbank von Android-Geräten. Für uns sind im Moment vor allem die beiden Konstanten `MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA` und `MediaStore.INTENT_ACTION_VIDEO_CAMERA` interessant. Sie werden nämlich als Actions für Intents verwendet, um die Kamera-App zu starten. Ausführliche Informationen bzgl. der Verwendung von Intents finden Sie in Kapitel 4, »Activities und Broadcast Receiver«.

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Benutzeroberfläche anzeigen
   setContentView(R.layout.main);
   Button foto = (Button) findViewById(R.id.foto);
   foto.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
         // Intent instanziieren
         Intent intent = new Intent(
               MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA);
         // Activity starten
         startActivity(intent);
```

```
      }
   });
   Button video = (Button) findViewById(R.id.video);
   video.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
         Intent intent = new Intent(
               MediaStore.INTENT_ACTION_VIDEO_CAMERA);
         startActivity(intent);
      }
   });
}
```

Listing 15.1 Auszug aus KameraDemo1.java

Eine auf diese Weise gestartete Kamera-App meldet allerdings keine Informationen an den Aufrufer zurück. Deshalb eignen sich die Intents `MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA` und `MediaStore.INTENT_ACTION_VIDEO_CAMERA` vor allem für Fire-and-forget-Szenarien. Damit ist gemeint, dass Sie dem Anwender Ihrer App zwar den Komfort bieten, ein Foto oder Video aufzunehmen, die auf diese Weise entstandenen Dateien aber nicht unmittelbar integrieren oder weiterverarbeiten. Wie Sie dies bewerkstelligen, zeige ich Ihnen im Folgenden.

Hinweis

Im Vergleich zu früheren Versionen des Android SDK ist die Kamerasimulation im Emulator etwas stabiler und greift soweit möglich auf mit Ihrem Entwicklungsrechner verbundene Webcams zu. Ich rate Ihnen dennoch, bei Ihren Tests echte Hardware verwenden.

15.1.2 Aufgenommene Fotos weiterverarbeiten

Um Daten von der Kamera-App in Ihrem Programm zu verwenden, rufen Sie `startActivityForResult()` auf. Das gesendete Intent enthält die Action `MediaStore.ACTION_IMAGE_CAPTURE`. Außerdem müssen Sie einen URI übergeben, der das aufgenommene Foto in der Mediendatenbank repräsentiert.

Wie das funktioniert, zeigt das Projekt *KameraDemo2*. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD. Nach dem Start sehen Sie einen, abgesehen von der Schaltfläche Foto aufnehmen, leeren Bildschirm. Nach einem Klick wird die Kamera-App gestartet. Wie Sie in Abbildung 15.4 sehen, erscheinen nach dem Auslösen die Buttons OK, Erneut aufnehmen und Abbrechen.

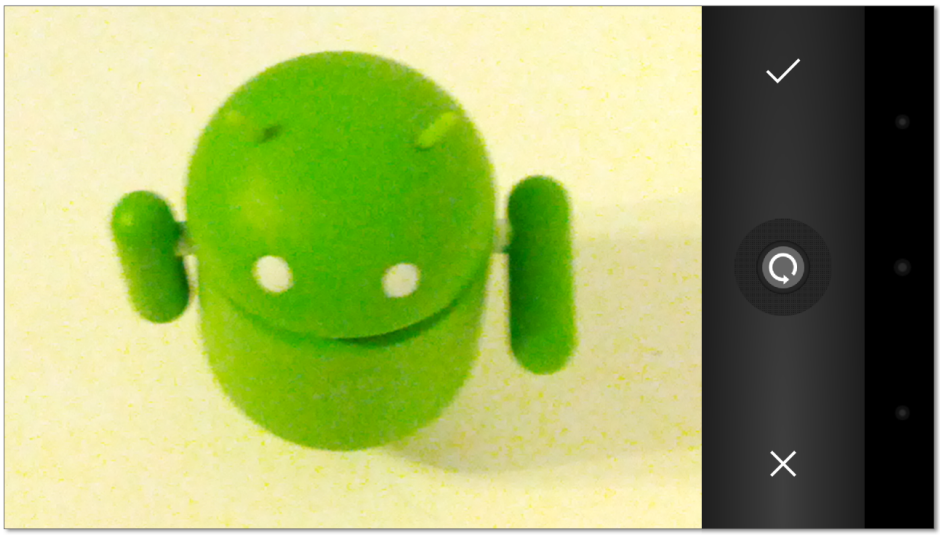


Abbildung 15.4 Die Kamera-App nach dem Auslösen

Hinweis

Beachten Sie, dass die Ausgestaltung der Benutzeroberfläche zwischen den verschiedenen Herstellern und dem Emulator unter Umständen variiert.

Mit OK gelangen Sie zu *KameraDemo2* zurück. Dort wird das aufgenommene Foto angezeigt. Wie das aussehen kann, sehen Sie in Abbildung 15.5.

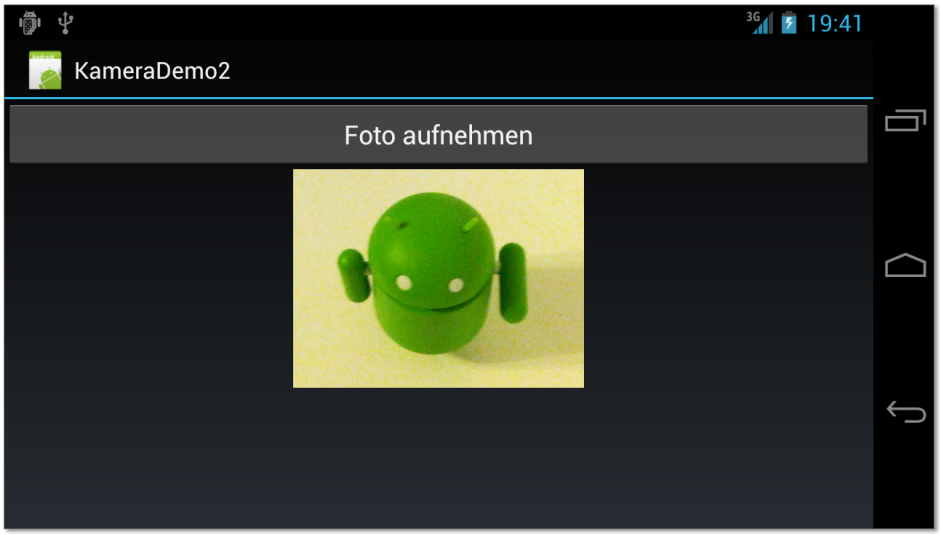


Abbildung 15.5 Das aufgenommene Foto in »KameraDemo2«

Die Klasse `KameraDemo2` leitet von `android.app.Activity` ab und überschreibt die beiden Methoden `onCreate()` und `onActivityResult()`. In `onCreate()` wird mit `setRequestedOrientation()` der Quermodus fest eingestellt und im Anschluss daran die Benutzeroberfläche angezeigt.

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   // Quermodus fest einstellen
   setRequestedOrientation(ActivityInfo.
         SCREEN_ORIENTATION_LANDSCAPE);
   // Benutzeroberfläche anzeigen
   setContentView(R.layout.main);
   imageView = (ImageView) findViewById(R.id.view);
   button = (Button) findViewById(R.id.shoot);
   button.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
         startCamera();
      }
   });
}
```

Listing 15.2 Auszug aus KameraDemo2.java

Klickt der Anwender auf die Schaltfläche FOTO AUFNEHMEN, wird die private Methode `startCamera()` aufgerufen. Diese erzeugt als Erstes einen neuen Eintrag in der systemweiten Mediendatenbank und feuert anschließend ein Intent mit der Action `MediaStore.ACTION_IMAGE_CAPTURE`. Der URI des angelegten Datensatzes wird dem Intent mit `putExtra()` übergeben.

```
private void startCamera() {
   ContentValues values = new ContentValues();
   values.put(MediaStore.Images.Media.TITLE,
         TITLE);
   values.put(MediaStore.Images.Media.DESCRIPTION,
         DESCRIPTION);
   values.put(MediaStore.Images.Media.MIME_TYPE,
         "image/jpeg");
   imageUri = getContentResolver().insert(
         MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
   Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
   intent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri);
```

```
   intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1);
   startActivityForResult(intent, IMAGE_CAPTURE);
}
```

Listing 15.3 Auszug aus KameraDemo2.java

Nach der Aufnahme wird die Methode `onActivityResult()` aufgerufen. Sie prüft zunächst den `requestCode`. Er muss dem bei `startActivityForResult()` übergebenen Wert entsprechen. Hat die Kamera-App als `resultCode RESULT_OK` gemeldet (der Benutzer hat ein Foto geschossen), ist alles in Ordnung, und das Foto kann angezeigt werden. Andernfalls muss der Eintrag mit `delete()` wieder aus der Mediendatenbank entfernt werden, weil der Anwender die Aufnahme abgebrochen hat.

```
protected void onActivityResult(int requestCode,
        int resultCode, Intent data) {
   if (requestCode == IMAGE_CAPTURE) {
      if (resultCode == RESULT_OK) {
         try {
               Bitmap b1 = MediaStore.Images.Media.getBitmap(
                     getContentResolver(), imageUri);
               // Größe des aufgenommenen Bildes
               float w1 = b1.getWidth();
               float h1 = b1.getHeight();
               // auf eine Höhe von 300 Pixel skalieren
               int h2 = 300;
               int w2 = (int) (w1 / h1 * (float) h2);
               Bitmap b2 = Bitmap.createScaledBitmap(
                                    b1, w2, h2, false);
               imageView.setImageBitmap(b2);
         } catch (FileNotFoundException e) {
            Log.e(TAG, "setBitmap()", e);
         } catch (IOException e) {
            Log.e(TAG, "setBitmap()", e);
         }
      } else {
         int rowsDeleted =
               getContentResolver().delete(imageUri, null, null);
         Log.d(TAG, rowsDeleted + " rows deleted");
      }
   }
}
```

Listing 15.4 Auszug aus KameraDemo2.java

Die Methode `MediaStore.Images.Media.getBitmap()` ist äußerst praktisch, um ein Bild, das in der Mediendatenbank gespeichert wurde, als Bitmap zur Verfügung zu stellen. Sie können Objekte des Typs `android.graphics.Bitmap` nämlich sehr einfach mit einem `ImageView` darstellen. Rufen Sie einfach die Methode `setImageBitmap()` einer `android.widget.ImageView`-Instanz auf. Allerdings können die von der Kamera gelieferten Fotos zu groß sein. Um eine Ausnahme zur Laufzeit zu vermeiden, sollten Sie das Bild wie im Beispiel gezeigt skalieren.

Hinweis

Um das Programm kurz zu halten, habe ich auf eine Überprüfung der Höhe des aufgenommenen Bildes verzichtet. Selbstverständlich ist eine Skalierung unnötig, wenn diese kleiner als die skalierte Höhe ist.

15.1.3 Mit der Galerie arbeiten

Die App *Galerie* zeigt alle Fotos und Videos der systemweiten Mediendatenbank und ist integraler Bestandteil der Plattform. Wie Sie sie in Ihre eigenen Programme integrieren, zeige ich Ihnen anhand des Projekts *GalleryDemo*, das Sie im Verzeichnis *Quelltexte* der Begleit-DVD finden. Unmittelbar nach dem Start ruft die App die Auswahlseite der Galerie auf, die Sie in Abbildung 15.6 sehen.

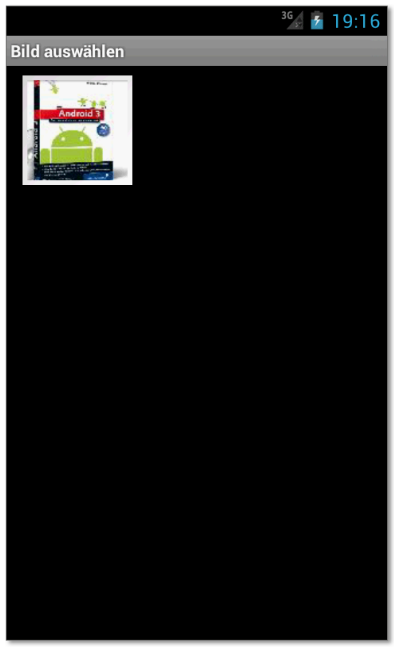


Abbildung 15.6 Die Auswahlseite der App »Galerie«

Tippt der Benutzer ein Bild an, ruft *GalleryDemo* abermals die Galerie auf. Diese zeigt die ausgewählte Datei in einer Art Detailansicht an, die in Abbildung 15.7 dargestellt ist. Die Klasse `GalleryDemo` ruft eine Auswahl auf, die alle Dateien im Bilderverzeichnis des externen Mediums anzeigt. Nachdem der Benutzer ein Bild ausgewählt hat, wird es in einer Detailansicht dargestellt.

```
public class GalleryDemo extends Activity {
   private static final int RQ_GALLERY_PICK = 1;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Intent intent = new Intent(Intent.ACTION_PICK,
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
      startActivityForResult(intent, RQ_GALLERY_PICK);
   }

   @Override
   protected void onActivityResult(int requestCode,
         int resultCode, Intent data) {
      super.onActivityResult(requestCode, resultCode, data);
      if (requestCode == RQ_GALLERY_PICK) {
         if (resultCode == RESULT_OK) {
            if (data != null) {
               Uri uri = data.getData();
               Intent i2 = new Intent(Intent.ACTION_VIEW, uri);
               startActivity(i2);
            }
         } else {
            finish();
         }
      }
   }
}
```

Listing 15.5 Auszug aus GalleryDemo.java

Der URI der Datei, die der Benutzer auf der Auswahlseite angeklickt hat, wird in den Extras eines Intents an `onActivityResult()` übermittelt. Er kann mit `getData()` abgefragt werden. Um keine Fehler zur Laufzeit zu riskieren, sollten Sie aber auf jeden Fall durch eine entsprechende `if`-Abfrage sicherstellen, dass nicht anstelle eines Intents `null` übergeben wurde.

Abbildung 15.7 Anzeige einer Datei in der »Galerie«

Um ein Bild anzuzeigen, packen Sie einfach dessen URI in ein Intent mit der Action `Intent.ACTION_VIEW` und rufen anschließend `startActivity()` auf.

Hinweis

Das Aussehen der Galerie-App variiert auf diversen Android-Geräten und dem Emulator.

15.1.4 Die Kamera-App erweitern

In meinen bisherigen Beispielen habe ich Ihnen gezeigt, wie Sie von Ihrem Programm aus Activities zum Aufnehmen von Fotos und Videos starten können. In diesem Abschnitt erweitern wir stattdessen die Kamera-App selbst. Diese bietet nämlich die Möglichkeit, eine Aufnahme an eine andere Activity weiterzuleiten.

Die Idee ist, Fotos per Bluetooth, MMS oder E-Mail versenden oder auf eine Website wie Picasa oder Facebook hochladen zu können. Aber wer sagt denn, dass man Fotos »nur« verschicken kann? Wie wäre es mit einer App, die unterschiedliche Filter anbietet, um ein Foto beispielsweise in Graustufen umzuwandeln, zu schärfen oder weichzuzeichnen? Das Projekt *ShareViaDemo* zeigt Ihnen, wie Sie hierzu vorgehen. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD.

Wenn Sie das Programm starten, tut sich scheinbar nichts. Ich habe nämlich keine Hauptaktivität implementiert. Deshalb erscheint die App nicht im Programmstarter. Rufen Sie die *Galerie* auf, wählen Sie ein Bild aus, und klicken Sie anschließend auf WEITERGEBEN bzw. TEILEN. Daraufhin öffnet sich ein Menü mit den möglichen »Empfängern«. Wie das im Emulator aussieht, ist in Abbildung 15.8 dargestellt. Echte Hardware zeigt das Ganze optisch reizvoller an, die Funktionalität bleibt aber gleich. Klicken Sie auf DEMO SENDER. Das von Ihnen ausgewählte Foto wird nun in Graustufen angezeigt.

Die Activity `DemoSender` prüft in ihrer Methode `onCreate()` als Erstes, ob ihr ein Intent übergeben wurde und ob dies die Action `Intent.ACTION_SEND` enthält. Ist dies nicht der Fall, beendet sie sich. Ansonsten wird die Benutzeroberfläche angezeigt und das als URI übergebene Bild geladen. Hierfür verwende ich die Ihnen bereits bekannte Methode `MediaStore.Images.Media.getBitmap()`.

Hinweis

Leider führt der Aufruf der Methode `getBitmap()` bei großen Dateien gelegentlich zum `OutOfMemoryError`. Um den Speicherhaushalt der virtuellen Maschine, in der Ihre App ausgeführt wird, nicht ins Wanken zu bringen, können Sie Referenzen auf nicht mehr benötigte Bitmaps durch Zuweisen von `null` als ungültig markieren. Vorher sollten Sie aber auf jeden Fall die Methode `recycle()` aufrufen. Aus Gründen der leichteren Nachvollziehbarkeit habe ich in meinen Beispielen auf solche Maßnahmen verzichtet.

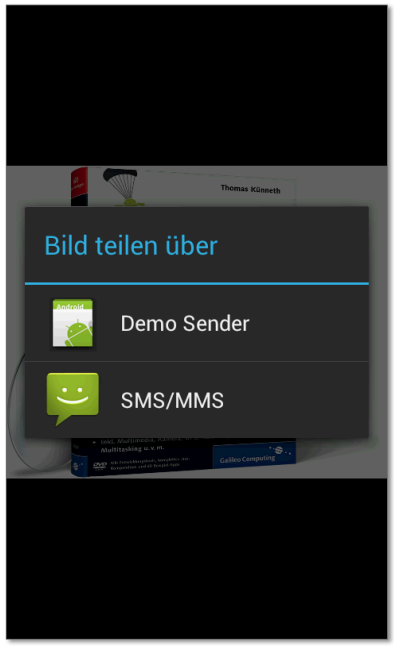


Abbildung 15.8 Das Menü »Weitergeben«

```
@Override
public void onCreate(Bundle state) {
   super.onCreate(state);
   // Wurde ein Intent übermittelt?
   Intent intent = getIntent();
   if (intent != null) {
      if (Intent.ACTION_SEND.equals(intent.getAction())) {
         // ja, dann Benutzeroberfläche anzeigen
         setContentView(R.layout.main);
         ImageView imageView =
               (ImageView) findViewById(R.id.image);
         // URI des Bildes
         final Uri imageUri =
               (Uri) intent.getExtras().get(Intent.EXTRA_STREAM);
         // Button
         final Button button
                        = (Button) findViewById(R.id.button);
         button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
               share(imageUri);
            }
         });
         try {
            // Bitmap erzeugen
            Bitmap bm1 = MediaStore.Images.Media.getBitmap(
                  getContentResolver(), imageUri);
            // in Graustufen umwandeln
            Bitmap bm2 = toGrayscale(bm1);
            // und anzeigen
            imageView.setImageBitmap(bm2);
         } catch (FileNotFoundException e) {
            Log.e(TAG, e.getClass().getSimpleName(), e);
         } catch (IOException e) {
            Log.e(TAG, e.getClass().getSimpleName(), e);
         }
      }
   } else {
      // nein, dann beenden
      finish();
   }
}
```

Listing 15.6 Auszug aus DemoSender.java

Das Umwandeln in Graustufen findet in der privaten Methode `toGrayscale()` statt. Als Erstes ermittle ich die Breite und Höhe des Quellbildes und erzeuge anschließend ein Objekt des Typs `android.graphics.Bitmap`, das die Pixeldaten der Graustufenversion enthalten wird.

Das `Canvas`-Objekt wird für den Kopiervorgang benötigt. Vereinfacht ausgedrückt »malt« `drawBitmap()` die alte Bitmap in die neue. Dass aus Farben Graustufen werden, ermöglicht die durch Aufruf von `setSaturation()` entsprechend konfigurierte `ColorMatrix`.

```
private Bitmap toGrayscale(Bitmap src) {
   // Breite und Höhe
   int width = src.getWidth();
   int height = src.getHeight();
   // neue Bitmap erzeugen
   Bitmap desti = Bitmap.createBitmap(width, height, Bitmap.Config.RGB_565);
   Canvas c = new Canvas(desti);
   Paint paint = new Paint();
   // Umwandlung in Graustufen
   ColorMatrix cm = new ColorMatrix();
   cm.setSaturation(0);
   ColorMatrixColorFilter f = new ColorMatrixColorFilter(cm);
   paint.setColorFilter(f);
   // mit Filter kopieren
   c.drawBitmap(src, 0, 0, paint);
   return desti;
}
```

Listing 15.7 Auszug aus DemoSender.java

Damit die Kamera- und Galerie-Apps die Klasse `DemoSender` als möglichen Empfänger von Bildern erkennen, muss in der Manifestdatei ein Intent-Filter für die Action `android.intent.action.SEND` konfiguriert werden. Dessen `<data />`-Element schränkt den MIME-Typ auf Bilder ein.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.thomaskuenneth.shareviademo"
      android:versionCode="1"
      android:versionName="1.0">
   <uses-sdk android:minSdkVersion="9" />
   <application android:icon="@drawable/icon"
         android:label="@string/app_name">
      <activity android:name=".DemoSender"
         android:label="@string/menu">
```

```
            <intent-filter>
               <action android:name="android.intent.action.SEND" />
               <category
                  android:name="android.intent.category.DEFAULT" />
               <data android:mimeType="image/*" />
            </intent-filter>
         </activity>
      </application>
</manifest>
```

Listing 15.8 Die Manifestdatei der App »ShareViaDemo«

Vielleicht fragen Sie sich, was nach dem Anwenden des Graustufenfilters und dem Anzeigen des Bildes passieren sollte. Die Benutzeroberfläche von *ShareViaDemo* enthält die Schaltfläche FERTIG. Wird sie angeklickt, sucht das Programm nach Activities, die die Action `Intent.ACTION_SEND` verarbeiten können und zeigt sie in einer Liste an.

Es handelt sich hierbei – Sie erinnern sich sicher – genau um diejenige Action, die die Klasse `DemoSender` selbst verarbeitet. Mit der statischen Methode `createChooser()` der Klasse `android.content.Intent` wird ein Intent erzeugt, das nach dem Aufruf von `startActivity()` eine Activity-Auswahl auf dem Bildschirm anzeigt. Ausführliche Informationen hierzu finden Sie in Kapitel 4, »Activities und Broadcast Receiver«.

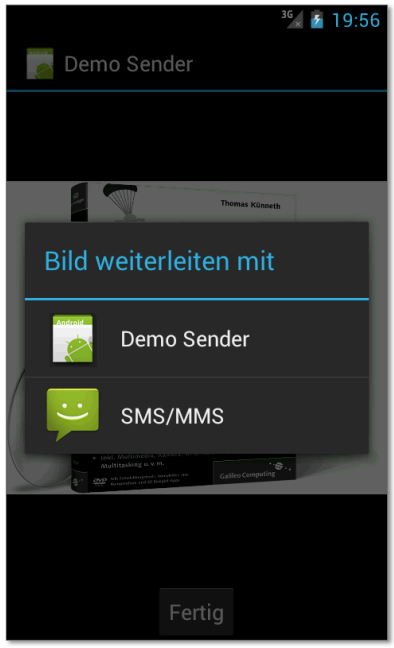


Abbildung 15.9 Die von »DemoSender« initiierte Empfängerauswahl

```
private void share(Uri uri) {
   // das zu feuernde Intent bauen
   Intent target = new Intent(Intent.ACTION_SEND, uri);
   target.setType("image/?");
   // Activity-Chooser-Intent bauen und feuern
   Intent intent = Intent.createChooser(target,
         getString(R.string.share_via));
   startActivity(intent);
}
```

Listing 15.9 Auszug aus DemoSender.java

Wenn die auf diese Weise gestartete Activity ihre Arbeit beendet hat, ist wieder *ShareViaDemo* zu sehen. Bei der Konzeption Ihrer App sollten Sie eine Schaltfläche BEENDEN in Erwägung ziehen, die die Activity ohne das Anzeigen der Auswahlliste schließt.

15.2 Die eigene Kamera-App

Sie haben gesehen, dass Android eine ganze Reihe von Bausteinen zur Verfügung stellt, um Apps mit einer Aufnahmefunktion für Bilder und Videos auszustatten. Bislang haben wir hierfür Teile der Android-eigenen Kamera-App verwendet. Wie Sie diese nachbauen, zeige ich Ihnen in diesem Abschnitt.

Schritt für Schritt lernen Sie in diesem Zusammenhang, wie eine Live-Vorschau programmiert wird, wie Sie zwischen den unterschiedlichen Kameras eines Android-Geräts umschalten und wie die eigentliche Aufnahme realisiert wird. Kopieren Sie hierzu das Projekt *KameraDemo3* aus dem Verzeichnis *Quelltexte* der Begleit-DVD in Ihren Eclipse-Arbeitsbereich, und importieren Sie es anschließend.

15.2.1 Live-Vorschau

Wir beginnen mit der Live-Vorschau. Das Display soll also kontinuierlich anzeigen, was das Objektiv gerade erfasst.

Die Benutzeroberfläche von *KameraDemo3* ist sehr einfach aufgebaut. Sie besteht aus einem `LinearLayout`, das eine View des Typs `android.view.SurfaceView` als einziges Kind enthält. Eine `SurfaceView` stellt einen dedizierten Zeichenbereich zur Verfügung, der zwar innerhalb einer View-Hierarchie angeordnet, aber von einem anderen Thread gezeichnet wird. Das hat den entscheidenden Vorteil, dass der Zeichen-Thread nicht auf die App warten muss, wenn diese mit anderen Aktionen beschäftigt ist.

Die Klasse android.view.SurfaceView

Der Zugriff auf die Oberfläche (engl. *Surface*) geschieht mittels Instanzen des Typs `android.view.SurfaceHolder`. Diese Klasse enthält unter anderem die beiden Methoden `addCallback()` und `removeCallback()`. Mit ihnen registrieren oder entfernen Sie ein Objekt des Typs `SurfaceHolder.Callback`. Dessen Methoden werden bei Änderungen an der Oberfläche aufgerufen. Welche dies sind, zeige ich Ihnen etwas später.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical"
>
   <SurfaceView
      android:id="@+id/view"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
   />
</LinearLayout>
```

Listing 15.10 Die Datei main.xml

Die Klasse `KameraDemo3` leitet von `android.app.Activity` ab und überschreibt die drei Methoden `onCreate()`, `onPause()` und `onResume()`. In `onCreate()` wird wie üblich die Benutzeroberfläche aufgebaut.

Um später auf die Oberfläche (surface) von `SurfaceView` zugreifen zu können, ermitteln Sie außerdem mit `getHolder()` ein Objekt des Typs `SurfaceHolder`. Dieses wird in `onResume()` und `onPause()` verwendet, um mit `addCallback()` eine Instanz der Klasse `SurfaceHolder.Callback` zu registrieren bzw. mit `removeCallback()` zu entfernen. Dieses Objekt ist übrigens die Activity selbst. `KameraDemo3` implementiert das Interface `SurfaceHolder.Callback`.

Hinweis

Auch wenn die Entwicklerdokumentation dies nicht so darstellt, ist der im folgenden Listing enthaltene Aufruf von `setType()` weiterhin erforderlich. Andernfalls drohen Abstürze bei `startPreview()`.

```
public class KameraDemo3 extends Activity implements SurfaceHolder.Callback {
   private static final String TAG = KameraDemo3.class.getSimpleName();
   private Camera camera = null;
   private SurfaceHolder holder = null;
```

```
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Benutzeroberfläche anzeigen
      setContentView(R.layout.main);
      final SurfaceView view = (SurfaceView) findViewById(R.id.view);
      holder = view.getHolder();
      holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
   }

   @Override
   protected void onPause() {
      super.onPause();
      // ganz wichtig: Callback entfernen und Kamera freigeben
      if (camera != null) {
         holder.removeCallback(this);
         camera.release();
         camera = null;
      }
   }

   @Override
   protected void onResume() {
      super.onResume();
      // prüfen, ob Kamera vorhanden ist
      camera = Camera.open();
      if (camera != null) {
         Camera.Parameters p = camera.getParameters();
         List<Camera.Size> list = p.getSupportedPreviewSizes();
         Camera.Size size = list.get(list.size() - 1);
         p.setPreviewSize(size.width, size.height);
         camera.setParameters(p);
         holder.addCallback(this);
      }
   }

   // Es folgen Methoden des Interfaces SurfaceHolder.Callback.
   ...
}
```

Listing 15.11 Auszug aus KameraDemo3.java

Vielleicht fragen Sie sich, wann ich (endlich) etwas über Kameras schreibe. Bislang haben Sie nur sehr viel über `SurfaceView` gelernt, aber das Ziel dieses Abschnitts ist

doch eigentlich die Implementierung einer Live-Vorschau, oder? Deswegen kommen wir jetzt zur Kamera.

Die Klasse android.hardware.Camera

Der Zugriff auf eine Kamera erfolgt über Instanzen des Typs `android.hardware.Camera`. In der Methode `onResume()` wird zuerst mit `Camera.open()` die Referenz auf ein solches Objekt ermittelt und der Variablen `camera` zugewiesen. Anschließend frage ich mit `getSupportedPreviewSizes()` ab, welche Vorschaugrößen die Kamera anbietet, und setze die kleinstmögliche mit `setPreviewSize()`.

Hinweis

Wenn Sie *KameraDemo3* starten, fällt Ihnen sicher auf, dass die Live-Vorschau mit Ausnahme der Titelzeile den gesamten Bildschirm einnimmt und sehr wahrscheinlich leicht verzerrt ist. Das Setzen der Vorschaugröße mit `setPreviewSize()` scheint also keine Wirkung zu haben.

Tatsächlich stellen Sie mit diesem Aufruf ein, in welcher Größe Android die Daten liefert. Dies hat auf die Größe einer View zunächst keine Auswirkung. Um die Demo übersichtlich zu halten, ist die Benutzeroberfläche, wie Sie gesehen haben, sehr einfach aufgebaut und ignoriert diesen Sachverhalt einfach.

In Ihrer eigenen App müssen Sie die Größe der SurfaceView so anpassen, dass sie dem Abbildungsverhältnis der Kameravorschau entspricht. Dies lässt sich recht einfach bewerkstelligen, indem Sie die View erst zur Laufzeit erzeugen. Ausführliche Informationen hierzu finden Sie in Kapitel 5, »Benutzeroberflächen«.

Da sich mehrere Anwendungen eine Kamera teilen, ist es unerlässlich, beim Verlassen der App belegte Ressourcen wieder freizugeben. Dies geschieht in `onPause()` durch Aufruf der Methode `release()`. Nun haben wir fast alle Puzzleteile beisammen. Offen ist nur noch die Frage, wohin die Kamera ihre Vorschaudaten übermitteln soll.

Etwas weiter zurück habe ich erklärt, dass das mit `addCallback()` registrierte `SurfaceHolder.Callback`-Objekt bei Änderungen an der Oberfläche informiert wird. Welche dies sind, hatte ich offen gelassen. Surfaces (also die Zeichenflächen von SurfaceViews) können erzeugt, geändert und zerstört werden.

Zu jedem dieser drei Ereignisse gibt es eine entsprechende Methode. Wir nutzen `surfaceCreated()`, um der Kamera mit `setPreviewDisplay()` die für die Live-Vorschau zu verwendende Oberfläche mitzuteilen. Außerdem starten wir mit `startPreview()` die Vorschau. Beides darf erst ausgeführt werden, nachdem ein Surface für das Zeichnen der Vorschau zur Verfügung steht.

```
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
   camera.stopPreview();
}
@Override
public void surfaceCreated(SurfaceHolder holder) {
   try {
      camera.setPreviewDisplay(holder);
      camera.startPreview();
   } catch (IOException e) {
      Log.e(TAG, "surfaceCreated()", e);
   }
}

@Override
public void surfaceChanged(SurfaceHolder holder,
      int format, int width,
      int height) {
}
```

Listing 15.12 Auszug aus KameraDemo3.java

Damit ist die Implementierung der Live-Vorschau fast abgeschlossen. Wir müssen nur noch in der Manifestdatei die Berechtigung `android.permission.CAMERA` anfordern, damit die App auf die eingebaute Kamera zugreifen darf. Natürlich fehlen zu einer »richtigen« Kamera noch ein paar Funktionen. Diese werde ich in den folgenden Abschnitten nachreichen.

Tipp

Wenn Sie Ihre App nicht nur für den Eigengebrauch entwickeln, sondern über *Google Play* vertreiben möchten, ist es wichtig, in der Manifestdatei zu vermerken, dass Ihr Programm zwingend eine Kamera voraussetzt. Fügen Sie einfach die Zeile

```
<uses-feature android:name="android.hardware.camera"
    android:required="true" />
```

hinzu. Ihre App wird dann nur auf Geräten mit eingebauter Kamera zum Download angeboten. Dies bewahrt Anwender vor Frust und schützt Sie vor unnötigen schlechten Kommentaren.

15.2.2 Kamera auswählen

Die App *KameraDemo3* zeigt stets die Live-Vorschau der Kamera auf der Rückseite eines Geräts, selbst wenn dieses auch eine Frontkamera zur Verfügung stellt. Seit Android 2.3 (Gingerbread) kennt die Klasse `android.hardware.Camera` die beiden Methoden `getNumberOfCameras()` und `getCameraInfo()`. Die erste liefert die Anzahl der vorhandenen Kameras. Mit der zweiten können Sie bestimmte Daten zu einer Kamera ermitteln.

Diese werden in einem Objekt des Typs `android.hardware.Camera.CameraInfo` abgelegt. Das folgende Quelltextfragment demonstriert die Vorgehensweise. Außerdem bereitet es die Frontkamera für die Benutzung vor. Ist keine vorhanden, wird stattdessen die rückwärtige geöffnet.

```
// In diesem Objekt werden Kamerainfos abgelegt.
Camera.CameraInfo cameraInfo = new CameraInfo();

// Anzahl der vorhandenen Kameras ermitteln
int numCams = Camera.getNumberOfCameras();
int front = -1;
for (int i = 0; i < numCams; i++) {
   // Infos zu Kamera Nr. I holen
   Camera.getCameraInfo(i, cameraInfo);

   // Position und Ausrichtung der Kamera ausgeben
   Log.d(TAG,
         (cameraInfo.facing == CameraInfo.CAMERA_FACING_BACK
                  ? "back"
                  : "front")
               + ", " + cameraInfo.orientation);
   if (cameraInfo.facing == CameraInfo.CAMERA_FACING_FRONT) {
      front = i;
   }
}

if (front != -1) {
   camera = Camera.open(front);
} else {
   camera = Camera.open();
}
```

Listing 15.13 Informationen zu den installierten Kameras ermitteln und ausgeben

Wenn Sie ein Android-Gerät mit zwei Kameras besitzen, können Sie das Quelltextfragment in die Klasse `KameraDemo3` übernehmen. In der Methode `onResume()` tauschen Sie einfach die Zeile

```
camera = Camera.open();
```

gegen das Listing aus. Selbstverständlich funktioniert das auch im Emulator. Allerdings kennt dieser nur die Kamera auf der Rückseite. In Abbildung 15.10 sind die im Emulator ermittelten Werte zu sehen. Der Wert 90 gibt an, um wie viel Grad Sie das Kamerabild im Uhrzeigersinn drehen müssen, damit es auf dem Gerätedisplay in dessen natürlicher Ausrichtung korrekt angezeigt wird.

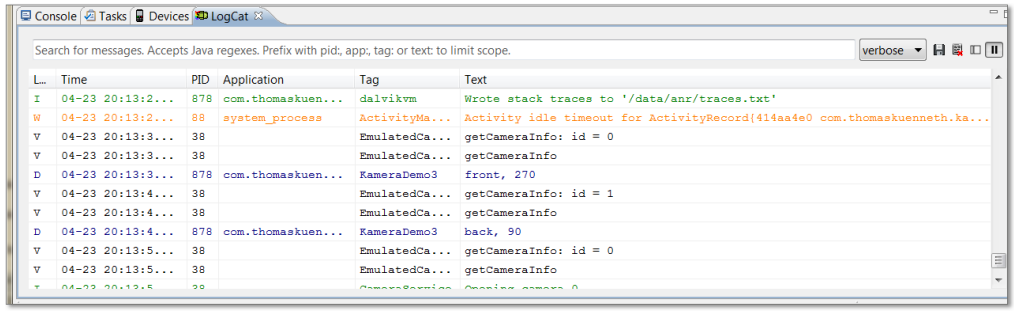


Abbildung 15.10 Ausgabe der Kamerainfos in der Sicht »LogCat«

Sicherlich fragen Sie sich, wie man eigentlich ein Foto aufnimmt. Eine Live-Vorschau ist zweifellos eine feine Sache, aber irgendwann möchte man schließlich den Auslöser drücken.

15.2.3 Fotos aufnehmen

Wie Sie aus Kapitel 10, »Das Android-Dateisystem«, wissen, ist die Berechtigung `android.permission.WRITE_EXTERNAL_STORAGE` nötig, um Dateien auf einem externen Medium ablegen zu können. Fügen Sie der Manifestdatei des Projekts *KameraDemo3* die Zeile

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

als Kind des `<manifest />`-Elements hinzu. Um eine Aufnahme auszulösen, soll der Anwender die Live-Vorschau antippen. Deshalb registrieren wir mit `setOnClickListener()` ein entsprechendes `OnClickListener`-Objekt. Ergänzen Sie hierzu die Methode `onCreate()` um das folgende Quelltextfragment. Fügen Sie es einfach vor der schließenden Klammer des Methodenrumpfes ein.

```
// Wir brauchen dieses Objekt für die Aufnahme.
final KameraCallbacks callbacks = new KameraCallbacks();
// auf Anklicken reagieren
view.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {
      camera.takePicture(callbacks, null, callbacks);
   }
});
```

Listing 15.14 KameraDemo3.java um eine Aufnahmefunktion erweitern

Hinweis

In vielen Beispielen findet sich unmittelbar nach `setOnClickListener()` zusätzlich der Methodenaufruf `setClickable(true)`. Dies geschieht aber automatisch, so dass dieses explizite »Anklickbarmachen« schlichtweg unnötig ist – und deshalb in meiner Implementierung auch nicht vorkommt.

Die von mir verwendete Variante der Methode `takePicture()` erwartet drei Parameter: ein Objekt des Typs `android.hardware.Camera.ShutterCallback` sowie zweimal `android.hardware.Camera.PictureCallback`. Diese beiden Interfaces werden durch die folgende Klasse `KameraCallbacks` implementiert:

```
package com.thomaskuenneth.kamerademo3;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.os.Environment;
import android.util.Log;

public class KameraCallbacks implements ShutterCallback, PictureCallback {
   private static final String TAG = KameraCallbacks.class.getSimpleName();

   @Override
   public void onShutter() {
      Log.d(TAG, "onShutter()");
   }
```

```
   @Override
   public void onPictureTaken(byte[] data, Camera camera) {
      Log.d(TAG, "onPictureTaken()");
      // In welchem Verzeichnis soll die Datei abgelegt werden?
      File dir = Environment.getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES);
      // ggf. Verzeichnisse anlegen
      dir.mkdirs();
      // Name der Datei
      File file = new File(dir, Long.toString(System.currentTimeMillis())
            + ".jpg");
      FileOutputStream fos = null;
      BufferedOutputStream bos = null;

      // Datei gepuffert schreiben
      try {
         fos = new FileOutputStream(file);
         bos = new BufferedOutputStream(fos);
         bos.write(data);
      } catch (IOException e) {
         Log.e(TAG, "onPictureTaken()", e);
      } finally {
         // Ströme schließen - etwaige Exceptions ignorieren
         if (bos != null) {
            try {
               bos.close();
            } catch (IOException e) {
            }
         }
         if (fos != null) {
            try {
               fos.close();
            } catch (IOException e) {
            }
         }
         // Live-Vorschau neu starten
         camera.startPreview();
      }
   }
}
```

Listing 15.15 Die Datei KameraCallbacks.java

Die Methode `onShutter()` wird im Moment der Aufnahme aufgerufen. Welche Aktionen Sie an dieser Stelle ausführen, hängt von der Art Ihrer Anwendung ab. Denkbar wäre, Zeitpunkt und Ort in einem internen Logbuch zu vermerken. `onPictureTaken()` kümmert sich um die Speicherung der angefallenen Bilddaten. Diese werden als `byte`-Feld übergeben, weil ihr Aufbau zum einen von Kameraeinstellungen, zum anderen aber auch vom gewählten Speicherformat abhängig ist. `takePicture()` sieht nämlich Callbacks für RAW- und JPG-Daten vor.

Meine Beispielimplementierung legt das aufgenommene Foto als JPG-Datei im Standardverzeichnis für Bilder auf dem externen Medium ab. Ausführliche Informationen zum Umgang mit Dateien und Verzeichnissen finden Sie in Kapitel 10, »Das Android-Dateisystem«.

Hinweis

Sie finden die um die Aufnahmefunktion erweiterte App als Projekt *KameraDemo3_full* im Verzeichnis *Quelltexte* der Begleit-DVD.

Um Ihre eigene Kamera-App mit Komfortfunktionen wie Zoom oder Weißabgleich auszustatten, sollten Sie sich in der Entwicklerdokumentation die Beschreibung der Klassen `android.hardware.Camera` und `android.hardware.Camera.Parameters` ansehen. Die Kamera bietet zahlreiche Einstellmöglichkeiten, die nicht nur die erzielbaren Ergebnisse, sondern auch die Größe der entstehenden Dateien beeinflussen.

15.3 Videos drehen

In diesem Abschnitt zeige ich Ihnen, wie Sie Videoclips aufzeichnen können. Einige der hier beschriebenen Vorgehensweisen werden Ihnen sicher bekannt vorkommen. Sie haben sie nämlich schon im Zusammenhang mit dem Bau einer eigenen Kamera-App kennengelernt.

15.3.1 Die App VideoCaptureDemo

Kopieren Sie das Projekt *VideoCaptureDemo* aus dem Verzeichnis *Quelltexte* der Begleit-DVD des Buches in Ihren Eclipse-Arbeitsbereich, und importieren Sie es anschließend. Grundsätzlich können Sie das Programm im Emulator ausprobieren. Allerdings ist dessen Unterstützung von Foto und Video derzeit leider noch sehr rudimentär. Wenn Ihnen ein Android-Gerät zur Verfügung steht, sollten Sie Ihre Experimente am besten mit echter Hardware durchführen.

Um Videos aufzeichnen zu können, müssen Apps die Berechtigungen `android.permission.CAMERA`, `android.permission.RECORD_AUDIO` und `android.permission.WRITE_`

EXTERNAL_STORAGE anfordern. Diese werden wie üblich in der Manifestdatei eingetragen:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.thomaskuenneth.videocapturedemo"
      android:versionCode="1"
      android:versionName="1.0">
   <uses-sdk android:minSdkVersion="9" />
   <uses-permission
      android:name="android.permission.RECORD_AUDIO" />
   <uses-permission android:name="android.permission.CAMERA" />
   <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <application android:icon="@drawable/icon"
         android:label="@string/app_name">
      <activity android:name=".VideoCaptureDemo"
            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 15.16 Manifestdatei der App »VideoCaptureDemo«

Rumpf der Klasse VideoCaptureDemo

Die Klasse VideoCaptureDemo leitet von android.app.Activity ab. In onCreate() wird als Erstes ein Objekt des Typs android.media.MediaRecorder instanziiert und anschließend durch Aufruf der privaten Methode initMedia-Recorder() initialisiert.

In einem weiteren Schritt wird die Benutzeroberfläche aus der Layoutdatei *main.xml* entfaltet und angezeigt. Tippt der Benutzer das einzige Element (ein Objekt des Typs android.view.SurfaceView) an, wird die Aufnahme gestartet oder gestoppt. Der aktuelle Modus findet sich in der Instanzvariablen isRecording.

```
package com.thomaskuenneth.videocapturedemo;

import java.io.File;
import java.io.IOException;
import android.app.Activity;
```

```
import android.media.CamcorderProfile;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnClickListener;

public class VideoCaptureDemo extends Activity implements
      SurfaceHolder.Callback {
   private static final String TAG = VideoCaptureDemo.class.getSimpleName();
   private MediaRecorder recorder;
   private SurfaceHolder surfaceHolder;
   private boolean isRecording;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // MediaRecorder erzeugen und initialisieren
      recorder = new MediaRecorder();
      initMediaRecorder();
      // Benutzeroberfläche anzeigen
      setContentView(R.layout.main);
      final SurfaceView view = (SurfaceView) findViewById(R.id.view);

      // Antippen startet und beendet die Aufnahme
      view.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View view) {
            if (isRecording) {
               recorder.stop();
               initMediaRecorder();
               prepareMediaRecorder();
            } else {
               recorder.start();
               isRecording = true;
            }
         }
      });
```

```
        // Surface-Holder ermitteln und Callback setzen
        surfaceHolder = view.getHolder();
        surfaceHolder.addCallback(this);
        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    // SurfaceHolder.Callback
    ...

    private void initMediaRecorder() {
        isRecording = false;
        recorder.setAudioSource(MediaRecorder.AudioSource.DEFAULT);
        recorder.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
        CamcorderProfile profile = CamcorderProfile
              .get(CamcorderProfile.QUALITY_HIGH);
        recorder.setProfile(profile);
        File dir = Environment
              .getExternalStoragePublicDirectory(Environment.DIRECTORY_MOVIES);
        dir.mkdirs();
        File file = new File(dir, "video.mp4");
        recorder.setOutputFile(file.getAbsolutePath());
        recorder.setMaxDuration(60000);
    }

    private void prepareMediaRecorder() {
        recorder.setPreviewDisplay(surfaceHolder.getSurface());
        try {
            recorder.prepare();
        } catch (IllegalStateException e) {
            Log.e(TAG, "prepareMediaRecorder()", e);
        } catch (IOException e) {
            Log.e(TAG, "prepareMediaRecorder()", e);
        }
    }
}
```

Listing 15.17 Auszug aus VideoCaptureDemo.java

Auch die Klasse `MediaRecorder` zeichnet die Vorschau während der Aufnahme in die Oberfläche (surface) eines `SurfaceView`. Aus diesem Grund muss wie in den vorangehenden Beispielen ein Objekt des Typs `SurfaceHolder.Callback` registriert werden. `VideoCaptureDemo` implementiert dieses Interface.

Implementierung von SurfaceHolder.Callback

Sobald die Oberfläche (surface) erzeugt wurde, wird in der privaten Methode `prepareMediaRecorder()` mit `setPreviewDisplay()` die Oberfläche für die Live-Vorschau gesetzt und anschließend mit `prepare()` das `MediaRecorder`-Objekt für die Aufnahme vorbereitet.

```
@Override
public void surfaceChanged(SurfaceHolder holder,
         int format, int width, int height) {
   Log.d(TAG, "surfaceChanged()");
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
   Log.d(TAG, "surfaceCreated()");
   prepareMediaRecorder();
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
   Log.d(TAG, "surfaceDestroyed()");
   if (isRecording) {
    recorder.stop();
    isRecording = false;
   }
   recorder.release();
   finish();
}
```

Listing 15.18 Auszug aus VideoCaptureDemo.java

Nach Änderungen an der Oberfläche (surface) wird von der Plattform die Methode `surfaceDestroyed()` aufgerufen. Meine Beispielimplementierung beendet in diesem Fall die Aufnahme, gibt die Ressourcen, die durch das `MediaRecorder`-Objekt belegt werden, frei und beendet anschließend mit `finish()` die Activity.

15.3.2 MediaRecorder konfigurieren

Die Klasse `MediaRecorder` bietet zahlreiche Konfigurationsmöglichkeiten. Diese beeinflussen nicht nur die Qualität des Audio- und Videosignals, sondern steuern auch den Aufnahmeverlauf. So können Sie durch Aufruf der Methode `setMaxDuration()` die maximale Länge einer Aufnahmesession in Millisekunden festlegen.

Die Methode `setMaxFileSize()` limitiert die erzeugte Datei auf eine bestimmte Größe in Byte.

In beiden Fällen sollten Sie eine Instanz des Typs `android.media.MediaRecorder.OnInfoListener` registrieren. Dessen Methode `onInfo()` wird aufgerufen, wenn eine der eben genannten Bedingungen eintritt.

```
recorder.setMaxDuration(60000);
recorder.setOnInfoListener(new OnInfoListener() {
   @Override
   public void onInfo(MediaRecorder mr, int what, int extra) {
      // auszuführende Aktion...
   }
});
```

Listing 15.19 Auf das Erreichen einer Aufnahmedauer reagieren

Kameraprofile sind eine weitere Möglichkeit, eine `MediaRecorder`-Instanz zu konfigurieren. Zunächst wird mit der statischen Methode `get()` der Klasse `android.media.CamcorderProfile` ein Basisprofil ermittelt und anschließend mit `setProfile()` in das `MediaRecorder`-Objekt übernommen.

Android stellt Ihnen als Entwickler mächtige Werkzeuge zur Verfügung, um leistungsfähige und innovative Multimedia-Apps zu entwickeln. Leider limitiert der Emulator aufgrund seiner nach wie vor rudimentären Unterstützung der entsprechenden Hardwarekomponenten die Testmöglichkeiten. Auch die Entwicklerdokumentation lässt in einigen Fällen Fragen offen. Aus diesem Grund finden Sie in der Lektüreliste im Anhang A Hinweise auf geeignete Spezialliteratur.

Kapitel 16
Kontakte und Organizer

Mit Smartphone und Tablet haben Sie jederzeit Zugriff auf Ihre Termine und Kontakte. In diesem Kapitel zeige ich Ihnen, wie Sie diese wertvollen Datenquellen mit Ihren eigenen Apps »anzapfen«.

Die hohe Kunst der App-Entwicklung liegt in der kreativen Kombination von vorhandener Hard- und Software zu etwas Neuem. Denken Sie an Apps, die Sensordaten mit Audio- und Videosignalen kombinieren und mit zusätzlichen Informationen aus dem Netz anreichern (Stichwort Augmented Reality). Für Sie als Entwickler gilt deshalb: Je mehr Datentöpfe Ihnen zur Verfügung stehen, desto größer sind Ihre Kombinationsmöglichkeiten.

Verglichen mit der Auswertung und Visualisierung von Ortsinformationen mag der Zugriff auf das Adressbuch oder den Kalender zunächst unspektakulär, vielleicht sogar langweilig wirken. Aber wäre es nicht toll, wenn Ihr Handy Sie bei einem eingehenden Anruf über anstehende Termine mit dem Gesprächspartner informieren oder an dessen Geburtstag erinnern würde? Oder stellen Sie sich eine App vor, die nach dem Anklicken der Notiz »Max Mustermann anrufen« eine Liste seiner Rufnummern einblendet und anbietet, automatisch zu wählen?

16.1 Kontakte

Android verteilt Kontaktdaten auf eine ganze Reihe von Tabellen, die Ihnen über einen Content Provider zur Verfügung stehen. Wie Sie diese Puzzleteile zusammensetzen müssen, ist in Googles Entwicklerdokumentationen leider nur recht oberflächlich beschrieben. Deshalb möchte ich Ihnen in den folgenden Abschnitten einige wichtige Zugriffstechniken vorstellen.

16.1.1 Eine einfache Kontaktliste ausgeben

Das Projekt *KontakteDemo1*, das Sie wie üblich im Verzeichnis *Quelltexte* der Begleit-DVD des Buches finden, gibt in der Sicht LogCat eine Liste der im Adressbuch gespeicherten Kontakte aus. Es greift auf einen Content Provider zu, dessen URI in der Konstante `ContactsContract.Contacts.CONTENT_URI` definiert ist.

Die Klasse `android.provider.ContactsContract` fungiert als Schnittstelle oder Vertrag zwischen Apps und dem Datenbestand. Dieser besteht aus drei Schichten, die sich in den Tabellen bzw. Klassen `ContactsContract.Data`,`ContactsContract.RawContacts` und `ContactsContract.Contacts` manifestieren. `Data` speichert beliebige persönliche Informationen wie Telefonnummer oder E-Mail-Adresse. `RawContacts` bündelt alle Informationen, die zu einer Person und einem Konto (zum Beispiel Twitter, Facebook oder Google Mail) gehören. `Contacts` schließlich fasst einen `RawContact` (oder mehrere) zu einem Gesamtkontakt zusammen.

Wie Sie aus Kapitel 12, »Content Provider«, wissen, erfolgt der Zugriff auf einen Content Provider über ein Objekt des Typs `android.content.ContentResolver`. Die Klasse `KontakteDemo1` ruft dessen Methode `query()` auf und iteriert über den zurückgelieferten `Cursor`. Damit das funktioniert, muss in der Manifestdatei die Berechtigung `android.permission.READ_CONTACTS` angefordert werden.

```
public class KontakteDemo1 extends Activity {
   private static final String TAG =
         KontakteDemo1.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      ContentResolver contentResolver = getContentResolver();

      // IDs und Namen aller sichtbaren Kontakte ermitteln
      String[] mainQueryProjection = {
            ContactsContract.Contacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME };
      String mainQuerySelection =
            ContactsContract.Contacts.IN_VISIBLE_GROUP
            + " = ?";
      String[] mainQuerySelectionArgs = new String[] { "1" };

      Cursor mainQueryCursor = contentResolver.query(
            ContactsContract.Contacts.CONTENT_URI,
            mainQueryProjection,
            mainQuerySelection, mainQuerySelectionArgs,
            null);

      // Trefferliste abarbeiten ...
      while (mainQueryCursor.moveToNext()) {
         String contactId = mainQueryCursor.getString(0);
         String displayName = mainQueryCursor.getString(1);
```

```
            Log.d(TAG, "===> "
                  + displayName
                  + " (" + contactId + ")");
        }
        mainQueryCursor.close();
    }
}
```

Listing 16.1 Auszug aus KontakteDemo1.java

Nach dem Start der App werden in der Sicht LogCat die Namen und IDs aller Kontakte ausgegeben. Wie das aussehen kann, ist in Abbildung 16.1 dargestellt.

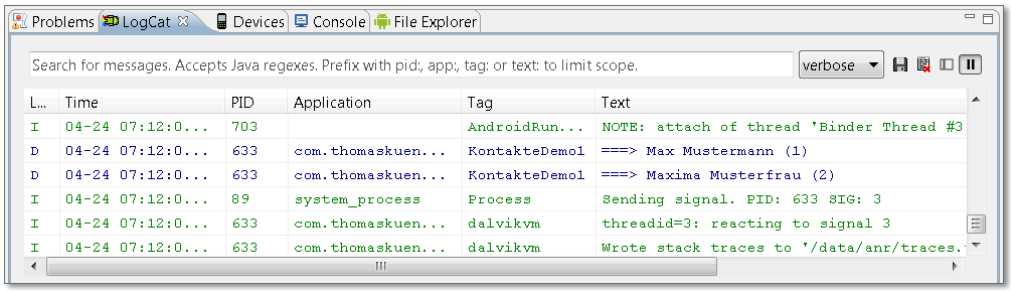


Abbildung 16.1 Ausgabe der App »KontakteDemo1«

Ist Ihnen aufgefallen, dass ich eine Auswahlbedingung definiert habe, die nur Einträge liefert, deren Tabellenspalte `IN_VISIBLE_GROUP` den Wert 1 enthält? Auf diese Weise erhalten Sie ausschließlich »richtige« Kontakte. Android merkt sich nämlich auch Absender von E-Mails. Diese würden ohne Verwendung der Bedingung ebenfalls geliefert, was in der Regel nicht wünschenswert ist. Im Emulator fällt dieses Verhalten sehr wahrscheinlich nicht auf, wohl aber auf echter Hardware.

`IN_VISIBLE_GROUP` wird in der Klasse `ContactsContract.Contacts` definiert. Viele weitere Konstanten sind dort nicht vorhanden. Interessante Informationen, wie etwa Geburtsdatum, E-Mail-Adresse oder Telefonnummer, müssen anderweitig ermittelt werden. Wie Sie hierzu vorgehen, zeige ich Ihnen anhand des Geburtsdatums.

16.1.2 Weitere Kontaktdaten ausgeben

Fügen Sie in der Klasse `KontakteDemo1` unterhalb der Zeile

```
Log.d(TAG, "===> " + displayName + " (" + contactId + ")");
```

die folgende Anweisung hinzu:

```
infosAuslesen(contentResolver, contactId);
```

Die Implementierung der neuen privaten Methode `infosAuslesen()` sehen Sie im folgenden Quelltextfragment. Sie finden die vollständige App auch als Projekt *KontakteDemo2* im Verzeichnis *Quelltexte* der Begleit-DVD. Kern ist auch hier der Aufruf von `query()`, wobei diesmal als URI `ContactsContract.Data.CONTENT_URI` übergeben wird.

Wie Sie bereits wissen, enthält diese Tabelle beliebige Einzelinformationen, unter anderem Jahres- und Geburtstage. Die Zuordnung zu einem Kontakt geschieht über die Spalte `CONTACT_ID`. Da wir eine solche ID als Parameter übergeben, können wir sehr einfach eine entsprechende Auswahlbedingung formulieren.

```
String dataQuerySelection = ContactsContract.Data.CONTACT_ID
      + " = ? AND " + ContactsContract.Data.MIMETYPE + " = ?";
```

Mit dem Teil nach `AND` wird die Treffermenge auf Ereignisse (`ContactsContract.CommonDataKinds.Event.CONTENT_ITEM_TYPE`) eingeschränkt.

```
private void infosAuslesen(ContentResolver contentResolver,
         String contactId) {
   String[] dataQueryProjection = new String[] {
         ContactsContract.CommonDataKinds.Event.TYPE,
         ContactsContract.CommonDataKinds.Event.START_DATE,
         ContactsContract.CommonDataKinds.Event.LABEL };
   String dataQuerySelection = ContactsContract.Data.CONTACT_ID
         + " = ? AND " + ContactsContract.Data.MIMETYPE + " = ?";
   String[] dataQuerySelectionArgs = new String[] {
         contactId,
         ContactsContract.CommonDataKinds.Event.CONTENT_ITEM_TYPE };
   Cursor dataQueryCursor = contentResolver.query(
         ContactsContract.Data.CONTENT_URI,
         dataQueryProjection,
         dataQuerySelection, dataQuerySelectionArgs,
         null);

   while (dataQueryCursor.moveToNext()) {
      int type = dataQueryCursor.getInt(0);
      String label = dataQueryCursor.getString(2);
      if (ContactsContract.CommonDataKinds.Event.TYPE_BIRTHDAY
            == type) {
         String stringBirthday = dataQueryCursor.getString(1);
         Log.d(TAG, "     birthday: "
               + stringBirthday);
      } else {
         String stringAnniversary = dataQueryCursor.getString(1);
```

```
            Log.d(TAG, "     event: " + stringAnniversary
                  + " (type="
                  + type + ", label=" + label + ")");

            if (ContactsContract.CommonDataKinds.Event.TYPE_ANNIVERSARY
               == type) {
               Log.d(TAG, "     TYPE_ANNIVERSARY");
            } else
                if (ContactsContract.CommonDataKinds.Event.TYPE_CUSTOM
                   == type) {
               Log.d(TAG, "     TYPE_CUSTOM");
            } else {
               Log.d(TAG, "     TYPE_OTHER");
            }
         }
      }
   }
```

Listing 16.2 Geburtstage und Jahrestage auslesen

Die Abfrage liefert die drei Spalten `Event.TYPE` (Ereignistyp), `Event.START_DATE` (Startdatum des Ereignisses) und `Event.LABEL` (eine Beschreibung). Leider ist das Format des Startdatums derzeit nicht fest vorgegeben. Das folgende Quelltextfragment liefert meiner Erfahrung nach aber sehr oft das gewünschte Ergebnis. Die Methode `getDateFromString1()` parst den ihr übergebenen String als Datum im Format 19700829. Zwischen Jahr und Monat sowie zwischen Monat und Tag können beliebige Zeichen stehen.


```
/**
 * Datum im Format jjjjmmtt, also 19700829
 */
public static final SimpleDateFormat FORMAT_YYYYMMDD
      = new SimpleDateFormat("yyyyMMdd");

public static Date getDateFromString1(String string) {
   Date result = null;

   if (string != null) {
      Pattern p = Pattern.compile(
            "(\\d\\d\\d\\d).*(\\d\\d).*(\\d\\d)",
            Pattern.DOTALL);
      Matcher m = p.matcher(string.subSequence(0,
                            string.length()));
      if (m.matches()) {
```

```
            String date = m.group(1) + m.group(2) + m.group(3);
            try {
               result = FORMAT_YYYYMMDD.parse(date);
            } catch (Throwable tr) {
               Log.e(TAG, "getDateFromString1()", tr);
            }
         }
      }
      return result;
   }
```

Listing 16.3 Datum von String nach Date wandeln

Haben Sie schon einmal versucht, im Emulator einem Kontakt ein Geburtsdatum oder einen Jahrestag zuzuweisen? Leider ist dies derzeit nicht möglich – zumindest, wenn Sie die Android-eigene Kontakte-App verwenden. Mit eigenen Programmen können Sie nämlich sehr wohl schreibend auf Kontaktdaten zugreifen. Wie das funktioniert, zeige ich Ihnen im folgenden Abschnitt.

16.1.3 Geburtstage hinzufügen und aktualisieren

Kopieren und importieren Sie das Projekt *KontakteDemo3* in Ihren Eclipse-Arbeitsbereich. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD. Die App sucht nach einem Kontakt, dessen angezeigter Name »Testperson« lautet. Wird ein solcher Datensatz gefunden, setzt das Programm den Geburtstag der Person auf das aktuelle Datum. War schon ein Eintrag vorhanden, wird das Geburtsjahr um 1 erniedrigt – der Kontakt wird also mit jedem Programmstart ein Jahr älter.

Wenn Sie die App ausführen, wird in der Sicht LogCat eine Meldung ausgegeben, dass die Testperson nicht gefunden wurde. Um dies zu korrigieren, legen Sie, wie in Abbildung 16.2 dargestellt, einen neuen Kontakt an. Ist die Testperson vorhanden, ermittelt das Programm die ID dieses Datensatzes. Wie Sie bereits wissen, wird diese für eine Suche in der Tabelle `Data` benötigt. Wurde dort schon ein Geburtstag eingetragen, aktualisieren wir diesen. Andernfalls wird ein neuer Datensatz hinzugefügt.

Diese Vorgehensweise können Sie in der Methode `updateOrInsertBirthday()` einfach nachvollziehen. Ein Geburtstag wurde schon eingetragen, wenn es in der Tabelle `Data` eine Zeile gibt, deren Spalte `CONTACT_ID` der übergebenen Kontakt-ID entspricht, `MIMETYPE` den Wert `Event.CONTENT_ITEM_TYPE` und `TYPE` den Wert `Event.TYPE_BIRTHDAY` hat. In diesem Fall muss nur das aktuelle Geburtsdatum ausgelesen und das Jahr um 1 verringert werden. `update()` schreibt das geänderte Attribut zurück in die Tabelle.

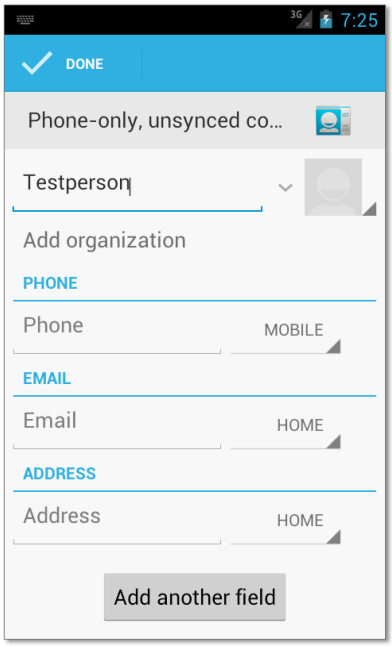


Abbildung 16.2 Einen Kontakt hinzufügen


```
public class KontakteDemo3 extends Activity {
   private static final String TAG
         = KontakteDemo3.class.getSimpleName();
   private static final SimpleDateFormat DATE_FORMAT
         = new SimpleDateFormat("yyyy-MM-dd");
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Content Resolver
      ContentResolver contentResolver = getContentResolver();

      // nach "Testperson" suchen
      String[] mainQueryProjection
            = { ContactsContract.Contacts._ID };
      String mainQuerySelection =
            ContactsContract.Contacts.IN_VISIBLE_GROUP
            + " = ?"
            + " AND "
            + ContactsContract.Contacts.DISPLAY_NAME
            + " is ?";
      String[] mainQuerySelectionArgs
            = new String[] { "1", "Testperson" };
```

```
      Cursor mainQueryCursor = contentResolver.query(
            ContactsContract.Contacts.CONTENT_URI,
            mainQueryProjection,
            mainQuerySelection, mainQuerySelectionArgs,
            null);

      if (mainQueryCursor != null) {
         if (mainQueryCursor.moveToNext()) {
            String contactId = mainQueryCursor.getString(0);
            Log.d(TAG, "===> Testperson gefunden"
                       + " (" + contactId + ")");
            updateOrInsertBirthday(contentResolver, contactId);
         } else {
            Log.d(TAG, "Testperson nicht gefunden");
         }
         mainQueryCursor.close();
      }
      // Activity beenden
      finish();
   }

   // updateOrInsertBirthday()
   ...
}
```

Listing 16.4 Auszug aus KontakteDemo3.java

Das Anlegen einer neuen Tabellenzeile folgt dem Ihnen aus Kapitel 12, »Content Provider«, bereits bekannten Schema. Die Methode `insert()` erhält ein `ContentValues`-Objekt, das mit Spalte-Wert-Paaren gefüllt wurde. Zu beachten ist allerdings, dass Sie im Gegensatz zu einem Update in der Tabelle `RawContacts` nach einer Zeile suchen müssen, die in der Spalte `CONTACT_ID` die übergebene Kontakt-ID enthält. Den Inhalt der Spalte `RawContacts._ID` müssen Sie mit der Anweisung

```
values.put(
   ContactsContract.CommonDataKinds.Event.
   RAW_CONTACT_ID, rawContactId);
```

in das `ContentValues`-Objekt übernehmen, sonst bricht `insert()` zur Laufzeit mit einer Ausnahme ab. Beachten Sie, dass Sie für das Ändern oder Hinzufügen von Kontaktdaten die Berechtigung `android.permission.WRITE_CONTACTS` in der Manifestdatei Ihrer App anfordern müssen.

```
private void updateOrInsertBirthday(
    ContentResolver contentResolver, String contactId) {
  String[] dataQueryProjection = new String[] {
      ContactsContract.CommonDataKinds.Event._ID,
      ContactsContract.CommonDataKinds.Event.START_DATE };
  String dataQuerySelection = ContactsContract.Data.CONTACT_ID
      + " = ? AND " + ContactsContract.Data.MIMETYPE
      + " = ?" + " AND "
      + ContactsContract.CommonDataKinds.Event.TYPE
      + " = ?";
  String[] dataQuerySelectionArgs = new String[] {
      contactId,
      ContactsContract.CommonDataKinds.Event.CONTENT_ITEM_TYPE,
      Integer.toString(
      ContactsContract.CommonDataKinds.Event.TYPE_BIRTHDAY) };

  // Gibt es einen Geburtstag zu Kontakt #contactId?
  Cursor dataQueryCursor = contentResolver.query(
      ContactsContract.Data.CONTENT_URI,
      dataQueryProjection,
      dataQuerySelection, dataQuerySelectionArgs, null);

  if (dataQueryCursor != null) {
    if (dataQueryCursor.moveToNext()) {
      // ja, Eintrag gefunden
      String dataId = dataQueryCursor.getString(0);
      String date = dataQueryCursor.getString(1);
      Log.d(TAG, "Geburtstag (_id=" + dataId + "): " + date);

      // Jahr um 1 verringern
      try {
         Date d = DATE_FORMAT.parse(date);
         Calendar cal = Calendar.getInstance();
         cal.setTime(d);
         cal.add(Calendar.YEAR, -1);
         d = cal.getTime();
         date = DATE_FORMAT.format(d);
         Log.d(TAG, "neues Geburtsdatum: " + date);
         // Tabelle aktualisieren
         String updateWhere =
             ContactsContract.CommonDataKinds.Event._ID
             + " = ?" + " AND "
             + ContactsContract.Data.MIMETYPE
```

```
            + " = ?" + " AND "
            + ContactsContract.CommonDataKinds.Event.TYPE
            + " = ?";
        String[] updateSelectionArgs = new String[] {
           dataId,
           ContactsContract.CommonDataKinds.Event.
             CONTENT_ITEM_TYPE,
           Integer.toString(ContactsContract.CommonDataKinds.
           Event.TYPE_BIRTHDAY)
        };

        ContentValues values = new ContentValues();
        values.put(
            ContactsContract.CommonDataKinds.Event.START_DATE,
              date);
        int numRows = contentResolver.update(
            ContactsContract.Data.CONTENT_URI, values,
              updateWhere, updateSelectionArgs);
        Log.d(TAG, "update() war "
            + ((numRows == 0) ? "nicht " : "")
            + "erfolgreich");
      } catch (ParseException e) {
        Log.e(TAG, date, e);
      }

    } else {
      Log.d(TAG, "keinen Geburtstag gefunden");

      // Strings für die Suche nach RawContacts
      String[] rawProjection = new String[] { RawContacts._ID };
      String rawSelection = RawContacts.CONTACT_ID
           + " = ?";
      String[] rawSelectionArgs
           = new String[] { contactId };

      // Werte für Tabellenzeile vorbereiten
      ContentValues values = new ContentValues();
      values.put(
           ContactsContract.CommonDataKinds.Event.START_DATE,
           DATE_FORMAT.format(new Date()));
      values.put(
         ContactsContract.Data.MIMETYPE,
         ContactsContract.CommonDataKinds.Event.CONTENT_ITEM_TYPE);
```

```
      values.put(ContactsContract.CommonDataKinds.Event.TYPE,
         ContactsContract.CommonDataKinds.Event.TYPE_BIRTHDAY);

      // alle RawContacts befüllen
      Cursor c = contentResolver.query(RawContacts.CONTENT_URI,
         rawProjection, rawSelection,
         rawSelectionArgs, null);
      while (c.moveToNext()) {
         String rawContactId = c.getString(0);
         values.put(
            ContactsContract.CommonDataKinds.Event.RAW_CONTACT_ID,
            rawContactId);
         Uri uri = contentResolver.insert(
             ContactsContract.Data.CONTENT_URI, values);
         Log.d(TAG,
             "   ---> Hinzufügen des Geburtstags "
             + "für RawContacts-Id "
             + rawContactId
             + " war"
             + ((uri == null)
                  ? " nicht erfolgreich"
                  : " erfolgreich"));
      }
      c.close();
    }
    dataQueryCursor.close();
  }
}
```

Listing 16.5 Auszug aus KontakteDemo3.java

Hinweis

Die Beziehungen zwischen den Tabellen der Kontaktdatenbank sind recht komplex. Deshalb sollten Sie Schreiboperationen sehr ausführlich im Emulator testen. Machen Sie vor Experimenten auf echter Hardware auf jeden Fall ein Backup Ihrer Kontakte. Eine kleine Unachtsamkeit bei der Entwicklung kann sonst zu ernsthaften Problemen führen.

Derzeit können mit der Android-eigenen Kontakte-App im Emulator keine Geburtstage erfasst werden können. Im Gegensatz zu früheren Versionen zeigt sie einen programmatisch (beispielsweise mit *KontakteDemo3*) hinzugefügten Geburtstag auch nicht mehr an. In Abbildung 16.3 sehen Sie deshalb die Detailseite eines Kontakts auf

echter Hardware. Alle für den Zugriff benötigten Mechanismen sind aber in der Plattform vorhanden und auch dokumentiert.

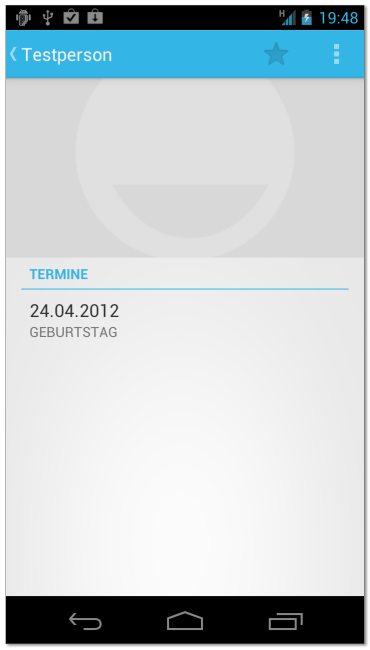


Abbildung 16.3 Die Kontakte-App zeigt den hinzugefügten Geburtstag an

16.2 Auf Google-Konten zugreifen

Wie Sie bereits wissen, kennt Android eine ganze Reihe von Content Providern. Mit der Anrufhistorie und Kontakten haben wir uns schon ausführlich beschäftigt. Woher diese ihre Daten beziehen, ist für Sie als Entwickler zweitrangig, da Sie über eine einheitliche Schnittstelle (einen Content Resolver) auf die tabellenartigen Daten zugreifen. Etwaige Anmeldevorgänge, Passwortabfragen und Synchronisierungen werden automatisch abgewickelt.

Auch wenn dies schon sehr lange nicht mehr zwingend notwendig ist, haben viele Benutzer von Android-Geräten ein Google-Konto. Es bietet Zugriff auf zahlreiche Dienste, beispielsweise Google Mail, Google Reader und Google Calendar. Nicht alle stehen unmittelbar über Content Provider zur Verfügung. In diesem Abschnitt möchte ich Ihnen deshalb zeigen, wie Sie auf Google-Konten und damit verbundene Dienste zugreifen können.

16.2.1 Emulator konfigurieren

Neue Konten werden über SETTINGS • ACCOUNTS & SYNC (bzw. EINSTELLUNGEN • KONTEN & SYNCHRONISIERUNG auf deutschsprachigen Geräten) eingerichtet. Die entsprechende Seite sehen Sie in Abbildung 16.4. Allerdings können mit den Standard-Emulator-Images nur Exchange-Konten hinzugefügt werden.

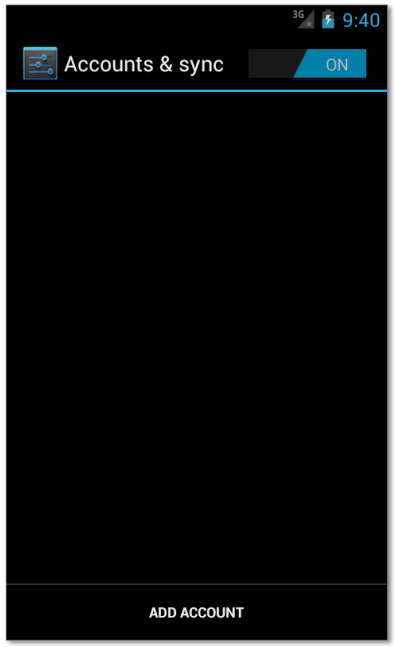


Abbildung 16.4 Hinzufügen eines neuen Kontos

Um die nachfolgenden Beispiele ausprobieren zu können, müssen Sie ein virtuelles Gerät auf Grundlage der Google APIs anlegen. Laden Sie im *Android SDK Manager* soweit noch nicht geschehen das Paket *Google APIs* herunter. In Abbildung 16.5 sehen Sie, welche Einstellungen im Dialog CREATE NEW ANDROID VIRTUAL DEVICE (AVD) vorzunehmen sind. Wichtig ist insbesondere, dass in der Klappliste TARGET ein mit GOOGLE APIS (GOOGLE INC.) beginnender Eintrag ausgewählt wird.

Klicken Sie auf CREATE AVD, um das virtuelle Gerät anzulegen und den Dialog zu schließen. Bevor Sie den Emulator starten, kopieren Sie bitte das Projekt *AccountDemo1* aus dem Verzeichnis *Quelltexte* der Begleit-DVD des Buches in Ihren Eclipse-Arbeitsbereich, und importieren Sie es anschließend.

Die Klasse `AccountDemo1` leitet von `android.app.Activity` ab und überschreibt nur die Methode `onCreate()`. Nachdem mit `AccountManager.get(this);` ein Objekt des Typs `android.accounts.AccountManager` ermittelt wurde, gibt das Programm die Zahl der gefundenen Konten sowie Informationen zu ihnen in der Sicht LOGCAT aus.

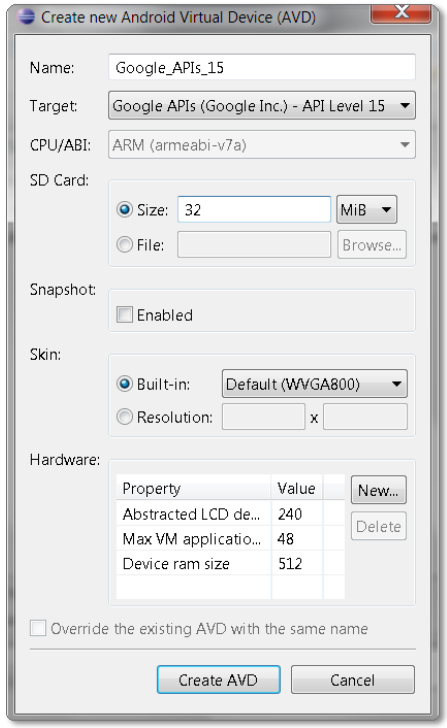


Abbildung 16.5 Ein AVD mit Google APIs anlegen

```
public class AccountDemo1 extends Activity {
   private static final String TAG
         = AccountDemo1.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      AccountManager am = AccountManager.get(this);
      Account[] accounts = am.getAccounts();
      Log.d(TAG, "Anzahl gefundener Konten: " + accounts.length);
      for (Account account : accounts) {
         Log.d(TAG, account.toString());
      }
      finish();
   }
}
```

Listing 16.6 Auszug aus AccountDemo1.java

Wenn Sie die App nun starten, werden keine Konten gefunden. Öffnen Sie deshalb die Einstellungsseite zum Anlegen von Konten, und fügen Sie ein Konto hinzu. Da Sie ein Google-APIs-Systemabbild verwenden, können Sie auf der in Abbildung 16.6 gezeigten Seite GOOGLE auswählen.

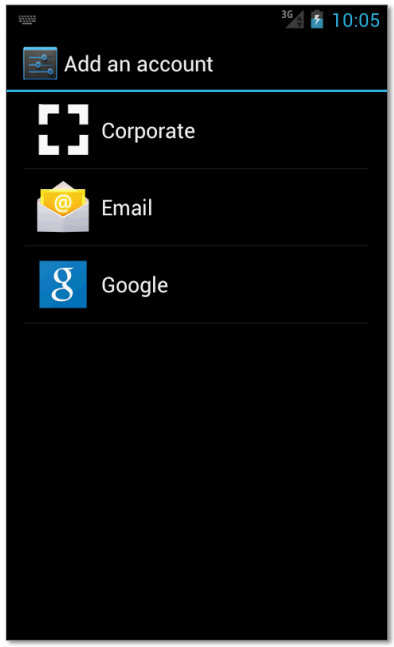


Abbildung 16.6 Auswahlseite mit hinzufügbaren Konten

Daraufhin wird ein Assistent zum Hinzufügen eines Google-Kontos gestartet. Sie müssen auswählen, ob Sie ein neues anlegen oder sich an einem bestehenden anmelden möchten. Ich empfehle Ihnen, die Anmeldedaten Ihres persönlichen Google-Kontos zu verwenden.

In diesem Fall müssen Sie auf der in Abbildung 16.7 gezeigten Seite Ihren Benutzernamen und Ihr Passwort eingeben. Nach dem Beenden des Assistenten gelangen Sie wieder zu der Hauptseite der Kontoeinstellungen. Dort wird ihr hinzugefügtes Google-Konto in einer Liste angezeigt. Starten Sie die App *AccountDemo1* erneut. In der Sicht LOGCAT sollte nun ihr Google-Konto gemeldet werden. Wie Sie darauf zugreifen, zeige ich Ihnen im folgenden Abschnitt.

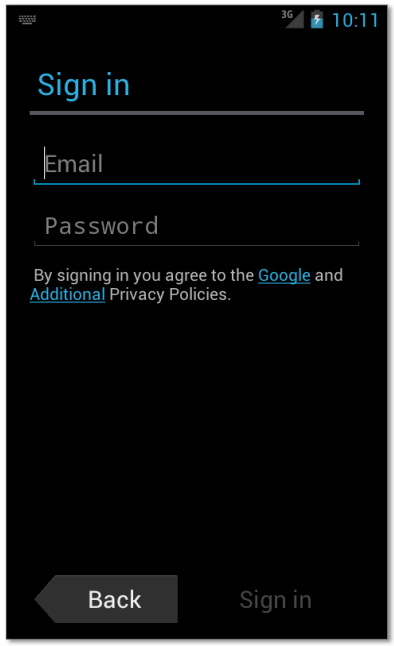


Abbildung 16.7 Anmeldung an einem Google-Konto

16.2.2 Aufgabenliste auslesen

Die Klasse `android.accounts.AccountManager`, mit der Sie im vorherigen Abschnitt schon kurz Bekanntschaft gemacht haben, bildet die Schnittstelle zu einem systemweiten Repository von Online-Konten, zum Beispiel Facebook, Twitter, Exchange oder Google.

Der Vorteil für Sie als Entwickler ist, dass Sie sich nicht um das Abfragen von Benutzernamen oder Passwörtern kümmern müssen. Dies regelt Android für Sie. Möchte eine App auf ein bereits eingerichtetes Konto zugreifen, kann der Benutzer dies mit einem Klick zulassen oder ablehnen. `AccountManager` nutzt ein Plugin-Konzept, um die unterschiedlichen Konto-Typen zu unterstützten. Solche Module können auch von Drittanbietern implementiert werden.

Wie Sie auf diese Weise auf ein Google-Konto zugreifen und eine Liste von Aufgaben anzeigen, die Sie in Ihrem Google-Kalendar abgelegt haben, demonstriere ich Ihnen anhand des Projekts *AccountDemo2*. Sie finden es im Verzeichnis *Quelltexte* der Begleit-DVD.

Die Klasse `AccountDemo2` leitet von `android.app.Activity` ab und überschreibt wie üblich die Methode `onCreate()`. Sie implementiert ferner das Interface `android.accounts.AccountManagerCallback`. In `onCreate()` ermitteln wir als Erstes eine `AccountManager`-Instanz und holen mit `getAccountsByType()` ein Feld des Typs

`android.accounts.Account`. Der Typ eines Kontos wird über eine Zeichenkette identifiziert, die für Google-Konten `com.google` lautet.

Wenn kein Konto vom gewünschten Typ eingerichtet wurde, sollten Sie den Anwender informieren und anschließend die Activity beenden oder ihn durch Aufruf der Methode `addAccount()` zum Anlegen eines neuen Kontos auffordern. Wurden mehrere Konten gefunden, bitten Sie den Benutzer, das gewünschte auszuwählen. Um die Klasse `AccountDemo2` übersichtlich zu halten, tut sie dies nicht.

Wie Sie bereits wissen, verwaltet `AccountManager` zu jedem Online-Konto die User Credentials, also Benutzernamen und Passwort. Damit sich eine App bei einem Dienst anmelden und seine Funktionen nutzen kann, muss sie diese natürlich kennen. Deshalb ruft `AccountDemo2` die Methode `getAuthToken()` auf.

Viele Services unterstützen das Konzept von *Auth Tokens*. Damit ein Dienstaufruf authentifiziert ablaufen kann, ohne jedes Mal Benutzernamen und Passwort übertragen zu müssen, wird stattdessen das Auth Token transportiert. Es repräsentiert einen einmal angemeldeten Benutzer. Um es zu erzeugen, kann sehr wohl die Eingabe von User Credentials notwendig sein. Darum kümmert sich aber Android. Der entsprechende Anmeldedialog ist in Abbildung 16.8 zu sehen. Ihre App muss also keinen eigenen zur Verfügung stellen.

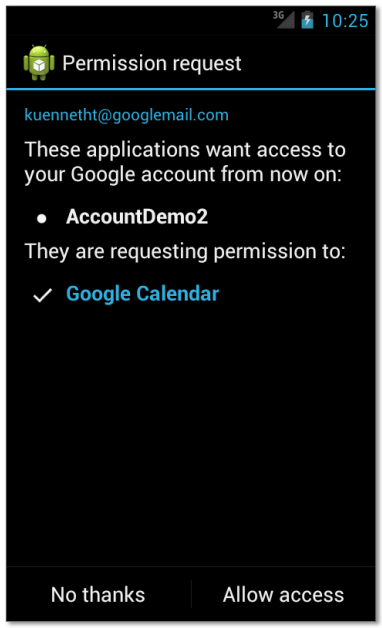


Abbildung 16.8 Abfrage, ob eine App auf ein Konto zugreifen darf

Die Bedeutung des Auth-Token-Typs ist vom verwendeten Kontotyp (beispielsweise `com.google`) abhängig, und Sie müssen ihn gegebenenfalls in der Entwicklerdokumen-

tation des Dienstanbieters nachlesen. Im Fall eines Google-Kontos spezifiziert der Auth-Token-Typ den zu nutzenden Service. `cl` repräsentiert den Google Calendar.[1]

Um das Ergebnis von `getAuthToken()` auszuwerten, rate ich Ihnen zur Nutzung von `AccountManagerCallback`. Dies mag auf den ersten Blick überraschen, da die Methode ja einen Rückgabewert hat. Bedenken Sie aber, dass dieser unter Umständen zunächst unvollständig sein kann, weil Android auf die Eingabe eines Passworts wartet. Bei der Verwendung eines Callbacks spielt die Frage nach dem Zeitpunkt der Fertigstellung keine Rolle.

In der Methode `run()` wird durch Aufruf von `getResult()` ein Objekt des Typs `android.os.Bundle` ermittelt. Das für die Kommunikation mit dem Server benötigte Auth Token erhalten wir mit `result.getString(AccountManager.KEY_AUTHTOKEN)`. Es wird neben der URL der herunterzuladenden Daten an die private Methode `getFromServer()` übergeben.

Beachten Sie das URL-Anhängsel `"&key=" + API_KEY`. Damit Ihre App auf Google-Tasks zugreifen darf, müssen Sie auf der Seite *http://code.google.com/apis/console* einen API-Schlüssel erzeugen. Die in Abbildung 16.9 gezeigte *Google APIs Console* kann hierzu unter API Access den benötigten Schlüssel *Simple API Access* generieren.

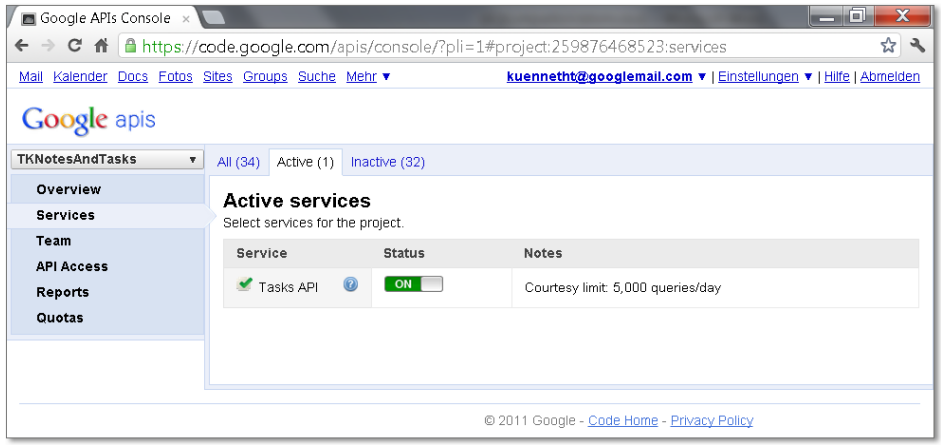


Abbildung 16.9 Google APIs Console

Die folgende Klasse zeigt alle Aufgaben in der Sicht *LogCat* an, die Sie in Ihrem Google Mail-Konto eingetragen haben:

```
public class AccountDemo2 extends Activity
      implements AccountManagerCallback<Bundle> {
   // Konto-Typ
   private static final String TYPE = "com.google";
```

1 *http://code.google.com/intl/de-DE/apis/gdata/faq.html#clientlogin*

```
// wird bei der Ermittlung des Auth Tokens benötigt
private static final String AUTH_TOKEN_TYPE = "cl";

// "Simple API Access" über https://code.google.com/apis/console
// generieren
private static final String API_KEY =
   "<Ihr Google API-Schlüssel>";

private static final String TAG =
      AccountDemo2.class.getSimpleName();
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   AccountManager accountManager = AccountManager.get(this);
   Account[] accounts = accountManager.getAccountsByType(TYPE);
   if (accounts.length == 1) {
      Bundle options = null;
      accountManager.getAuthToken(accounts[0],
            AUTH_TOKEN_TYPE, options,
            this, this, null);
   } else {
      finish();
   }
}

public void run(AccountManagerFuture<Bundle> future) {
   Bundle result;
   try {
      result = future.getResult();
      String token =
            result.getString(AccountManager.KEY_AUTHTOKEN);
      String tasks = getFromServer(
         "https://www.googleapis.com/tasks/v1/lists/
            @default/tasks?pp=1&key="
         + API_KEY, token);
      Log.d(TAG, tasks);
   } catch (OperationCanceledException e) {
      Log.e(TAG, "run()", e);
   } catch (AuthenticatorException e) {
      Log.e(TAG, "run()", e);
```

```
        } catch (IOException e) {
            Log.e(TAG, "run()", e);
        }
    }

    private String getFromServer(String url, String token) {
        StringBuilder sb = new StringBuilder();
        HttpGet get = new HttpGet(url);
        // Auth Token in den Header übertragen
        get.addHeader("Authorization", "GoogleLogin auth="
              + token);
        HttpClient httpclient = new DefaultHttpClient();
        HttpResponse r;
        try {
            // http-get
            r = httpclient.execute(get);
            InputStream is = r.getEntity().getContent();
            int ch;
            // Daten laden
            while ((ch = is.read()) != -1) {
                sb.append((char) ch);
            }
        } catch (ClientProtocolException e) {
            Log.e(TAG, "getFromServer()", e);
        } catch (IOException e) {
            Log.e(TAG, "getFromServer()", e);
        }
        return sb.toString();
    }
}
```

Listing 16.7 Auszug aus AccountDemo2.java

In `getFromServer()` wird das Auth Token einem `HttpGet`-Objekt durch Aufruf der Methode `addHeader()` übergeben. Beachten Sie, dass Sie die hier gezeigte Laderoutine für einen produktiven Einsatz unter anderem durch eine bessere Fehlerbehandlung absichern müssen. Für einfache Tests ist sie freilich ausreichend.

Hinweis

Damit der Zugriff auf die Aufgabenliste wie hier beschrieben klappt, muss Ihre App die Berechtigungen `android.permission.GET_ACCOUNTS`, `android.permission.USE_CREDENTIALS` und `android.permission.INTERNET` anfordern.

16.3 Kalender und Termine

Vor *Ice Cream Sandwich* bot Android erstaunlich wenige öffentliche und dokumentierte Schnittstellen für Apps, um Termine auszulesen oder zu bearbeiten. Glücklicherweise hat die Plattform seitdem deutlich dazu gelernt.

Hinweis

Derzeit ist der Emulator nur bedingt für den Test von kalenderbezogenen Apps geeignet, weil die Android-eigene Kalender-Anwendung nur in Verbindung mit einem Konto funktioniert.

16.3.1 Termine anlegen und auslesen

Das Projekt *KalenderDemo1* demonstriert, wie Sie durch das Versenden eines Intents mit der Standard-Kalender-App einen Termin anlegen können. Sie finden es wie üblich im Verzeichnis *Quelltexte* der Begleit-DVD des Buches. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich.

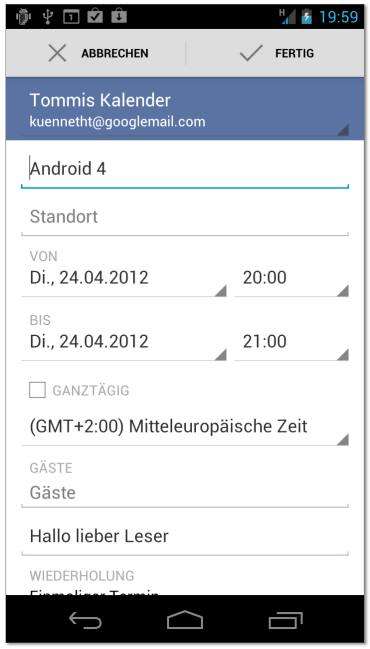


Abbildung 16.10 Eingabeseite für Termine

Nach dem Start erscheint die Eingabeseite für Termine. Sie sehen sie in Abbildung 16.10. Die Klasse `KalenderDemo1` leitet von `android.app.Activity` ab. In

`onCreate()` werden zunächst zwei `java.util.Date`-Objekte mit dem Beginn und dem Ende des anzulegenden Termins initialisiert.

Anschließend ruft die App die private Methode `createEntry()` auf. Diese erzeugt ein Intent mit der Action `Intent.ACTION_INSERT` und setzt mit `putExtra()` die Termin-Attribute Beginn, Ende, Titel und Beschreibung. `allDay` kennzeichnet einen Termin als ganztägig.

```
public class KalenderDemo1 extends Activity {

   private static final String TAG =
      KalenderDemo1.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // Beginn und Ende eines Termins
      Calendar cal = Calendar.getInstance();
      Date from = cal.getTime();
      cal.add(Calendar.HOUR_OF_DAY, 1);
      Date to = cal.getTime();
      // Termin anleen
      createEntry("Android 4", "Hallo lieber Leser",
                  from, to, false);
   }

   private void createEntry(String title,
                            String description,
                            Date from,
                            Date to,
                            boolean allDay) {
      Intent intent = new Intent(Intent.ACTION_INSERT,
                                 Events.CONTENT_URI);
      intent.putExtra(Events.TITLE, title);
      intent.putExtra(Events.DESCRIPTION, description);
      intent.putExtra(Events.DTSTART, from.getTime());
      intent.putExtra(Events.DTEND, to.getTime());
      intent.putExtra(Events.ALL_DAY, allDay);
      try {
         startActivity(intent);
```

```
      } catch (ActivityNotFoundException e) {
         Log.e(TAG, "keine passende Activity", e);
      }
   }
}
```

Listing 16.8 Auszug aus KalenderDemo1.java

16.3.2 Alarme verwalten

Würden Sie Ihre App gerne um eine Alarmfunktion erweitern? Mit Android 2.3 (Gingerbread) hat Google die Klasse `android.provider.AlarmClock` eingeführt. Sie enthält die Konstanten `ACTION_SET_ALARM`, `EXTRA_MESSAGE`, `EXTRA_HOUR` und `EXTRA_MINUTES`, mit denen Sie eine Activity zum Stellen eines Alarms starten können.

Dies kann – muss aber nicht – der standardmäßig mitgelieferte Wecker sein. Den entsprechenden Aufruf sehen Sie im folgenden Quelltextfragment. Damit er funktioniert, müssen Sie in der Manifestdatei die Berechtigung `com.android.alarm.permission.SET_ALARM` anfordern.

```
// einen neuen Alarm setzen
Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM);
intent.putExtra(AlarmClock.EXTRA_MESSAGE, "Hallo Android");
intent.putExtra(AlarmClock.EXTRA_HOUR, 20);
intent.putExtra(AlarmClock.EXTRA_MINUTES, 00);
startActivity(intent);
```

Listing 16.9 Einen Alarm setzen

In Honeycomb (Android 3.0) wurde die Konstante `EXTRA_SKIP_UI` hinzugefügt. Der Wert `true` signalisiert der App, die das Intent auswertet, dass sie keine Benutzeroberfläche darstellen, sondern das Ergebnis »nur« als Toast anzeigen und sich dann beenden soll. Bei `false` sind hingegen weitere Dialoge oder Rückfragen gestattet.

Wie Sie in Ihrer App auf die Action `ACTION_SET_ALARM` reagieren und somit einen eigenen Dialog zum Eingeben eines Alarms anzeigen können, demonstriere ich Ihnen anhand des Projekts *AlarmClockDemo1*, das Sie wie üblich im Verzeichnis *Quelltexte* der Begleit-DVD finden. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich.

Nach dem Start sehen Sie den in Abbildung 16.11 gezeigten Auswahldialog. Mit einem Klick auf AlarmClockDemo1 werden drei Werte in der Sicht LogCat ausgegeben. Dies sehen Sie in Abbildung 16.12. Anschließend beendet sich die App.

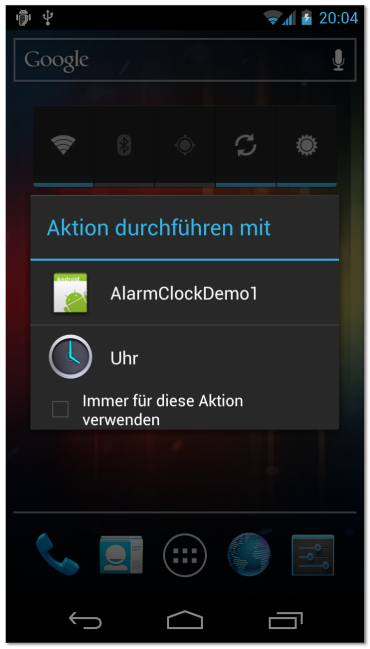


Abbildung 16.11 Dialog zum Auswählen einer Activity

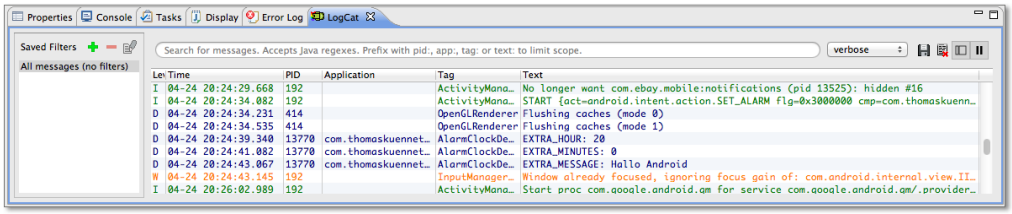


Abbildung 16.12 Ausgabe in der Sicht »LogCat«

Wie Sie aus Kapitel 4, »Activities und Broadcast Receiver«, wissen, können Activities gezielt auf Intents reagieren, indem Sie in der Manifestdatei einen Intent-Filter für die gewünschte Action definieren. Im Folgenden sehen Sie die Manifestdatei der App *AlarmClockDemo1*. Die Activity-Klasse `AlarmClockDemo1` erhält einen Intent-Filter für die Action `AlarmClock.ACTION_SET_ALARM`.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.thomaskuenneth.alarmclockdemo1"
      android:versionCode="1"
      android:versionName="1.0">
   <uses-sdk android:minSdkVersion="9" />
   <uses-permission
```

```
      android:name="com.android.alarm.permission.SET_ALARM" />
   <application android:icon="@drawable/icon"
      android:label="@string/app_name">
      <activity android:name=".AlarmClockDemo1"
            android:label="@string/app_name">

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

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

      </activity>
   </application>
</manifest>
```

Listing 16.10 Manifestdatei der App »AlarmClockDemo1«

Activities können mit `getIntent()` abfragen, welches Intent ihnen übermittelt wurde. Nach einer Prüfung auf `null` und einem Vergleich der gelieferten mit der erwarteten Action werden etwaige zusätzlich übertragene Parameter ausgewertet. Dies geschieht üblicherweise in `onCreate()`.

Die Klasse `AlarmClockDemo1` prüft, ob sie ein Intent mit der Action `AlarmClock.ACTION_SET_ALARM` erhalten hat. Ist dies der Fall, wird die übergebene Uhrzeit (Stunden und Minuten) sowie ein Nachrichtentext mit `getIntExtra()` bzw. `getStringExtra()` ausgelesen. Beachten Sie, dass diese Felder nicht vollständig gefüllt sein müssen.

```
public class AlarmClockDemo1 extends Activity {
   private static final String TAG =
         AlarmClockDemo1.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Intent i = getIntent();
      if ((i != null) && (AlarmClock.ACTION_SET_ALARM.equals(
            i.getAction()))) {
```

```
            Log.d(TAG,
                  "EXTRA_HOUR: "
                  + i.getIntExtra(AlarmClock.EXTRA_HOUR, -1));
            Log.d(TAG,
                  "EXTRA_MINUTES: "
                  + i.getIntExtra(AlarmClock.EXTRA_MINUTES, -1));
            Log.d(TAG,
                  "EXTRA_MESSAGE: "
                  + i.getStringExtra(AlarmClock.EXTRA_MESSAGE));
            // Alarm setzen
            ...
            // beenden
            finish();

         } else {
            // einen neuen Alarm setzen
            Intent intent = new Intent(AlarmClock.ACTION_SET_ALARM);
            intent.putExtra(AlarmClock.EXTRA_MESSAGE, "Hallo Android");
            intent.putExtra(AlarmClock.EXTRA_HOUR, 20);
            intent.putExtra(AlarmClock.EXTRA_MINUTES, 00);
            startActivity(intent);
            finish();
         }
      }
}
```

Listing 16.11 Auszug aus AlarmClockDemo.java

Vielleicht fragen Sie sich, warum Sie einen eigenen Dialog zur Eingabe von Alarmen anzeigen sollten. Dies bietet sich in erster Linie für Apps an, die den eingebauten Wecker vollständig ersetzen sollen. In diesem Fall müssen Sie Weckzeiten und Nachrichten in einer eigenen Datenbank verwalten. Ausführliche Informationen zu SQLite-Datenbanken finden Sie in Kapitel 11, »Datenbanken«.

Auch das Setzen der Alarme selbst liegt dann in Ihrer Verantwortung. Die Klasse `android.app.AlarmManager` stellt die Methode `set()` zur Verfügung. Sie feuert zu einem bestimmten Zeitpunkt ein `PendingIntent`. Mit ihm können Sie weitere Aktionen auslösen, zum Beispiel einen Text ausgeben oder einen Sound abspielen. Hierbei handelt es sich allerdings um fortgeschrittene Themen, die an dieser Stelle nicht weiter behandelt werden können.

16.3.3 Die Klasse CalendarContract

Wie Sie in den vorangehenden Kapiteln gesehen haben, spielen Content Provider eine äußerst wichtige Rolle in Android. Insofern liegt es nahe, dass die Plattform auch für Termine und Ereignisse auf dieses mächtige Instrument baut. Tatsächlich können Sie über einen Content Resolver Kalenderdaten ausgeben. Wie das funktioniert, zeigt die App *KalenderDemo2*. Sie finden das gleichnamige Projekt im Verzeichnis *Quelltexte* der Begleit-DVD. Kopieren und importieren Sie es in Ihren Eclipse-Arbeitsbereich.

Leider ist auch dieses Programm nur auf echter Hardware lauffähig. Die Klasse `KalenderDemo2` leitet von `android.app.Activity` ab. In der Methode `onCreate()` iteriert eine Schleife über einen von `query()` gelieferten Cursor. Die Spalten `_ID` und `TITLE` der gefunden Tabellenzeilen werden in der Sicht LogCat ausgegeben.

```
public class KalenderDemo2 extends Activity {

   private static final String TAG =
      KalenderDemo2.class.getSimpleName();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Cursor c = getContentResolver().query(Events.CONTENT_URI,
                                            null,
                                            null,
                                            null,
                                            null);
      if (c != null) {
         int indexId = c.getColumnIndex(Events._ID);
         int indexTitle = c.getColumnIndex(Events.TITLE);
         while (c.moveToNext()) {
            Log.d(TAG, "_ID: " + c.getString(indexId));
            Log.d(TAG, "TITLE: " + c.getString(indexTitle));
         }
         c.close();
      }
   }
}
```

Listing 16.12 Auszug aus KalenderDemo2.java

Um auf den Kalender zugreifen zu können, müssen Sie in der Manifestdatei die Berechtigung `android.permission.READ_CALENDAR` anfordern. Für schreibende Zugriffe ist zusätzlich `android.permission.WRITE_CALENDAR` erforderlich.

Anhang

A Literaturverzeichnis 427

B Die Begleit-DVD 429

C Häufig benötigte Code-Bausteine 435

Anhang A
Literaturverzeichnis

Bartosch, Oliver/Throll, Marcus: *Einstieg in SQL*. 4. Auflage. Galileo Press, 2010.

Bates, Bert/Sierra, Kathy: *Java von Kopf bis Fuß*. 1. Auflage. O'Reilly, 2006.

Beighley, Lynn: *SQL von Kopf bis Fuß*. 1. Auflage. O'Reilly, 2008.

Hunt, Andrew/Thomas, David: *Pragmatisch Programmieren*. *Unit-Tests mit JUnit*. 1. Auflage. Hanser, 2004.

Kreibich, Jay A.: *Using SQLite*. 1. Auflage. O'Reilly, 2010.

Künneth, Thomas/Wolf, Yvonne: *Einstieg in Eclipse 3.7*. 4. Auflage. Galileo Press, 2011.

Martin, Robert C.: *Clean Code*. *Refactoring, Patterns, Testen und Techniken für sauberen Code*. 1. Auflage. mitp, 2009.

Oechsle, Rainer: *Parallele und verteilte Anwendungen in Java*. 3., erweiterte Auflage. Hanser, 2011.

Ullenboom, Christian: *Java ist auch eine Insel*. *Das umfassende Handbuch*. 10. Auflage. Galileo Press, 2012.

Van Every, Shawn: *Pro Android Media*: *Developing Graphics, Music, Video, and Rich Media Apps for Smartphones and Tablets*. 1. Auflage. Apress, 2010.

Anhang B
Die Begleit-DVD

Neben Versionen des Android SDKs, des Java SDKs 6 und Eclipse Classic 3.7 sowie dem Kompendium *Java ist auch eine Insel* enthält die Begleit-DVD zum Buch über 60 Apps. Wenn Sie diese ausprobieren möchten, kopieren Sie den oder die gewünschten Ordner aus dem DVD-Verzeichnis *Quelltexte* in Ihren Eclipse-Arbeitsbereich.

Um dessen Speicherort zu ermitteln, klicken Sie in der Menüleiste von Eclipse auf File • Switch Workspace • Other. Den Pfad können Sie dem Eingabefeld Workspace des in Abbildung B.1 gezeigten Dialogs Workspace Launcher entnehmen. Mit Cancel schließen Sie den Dialog.

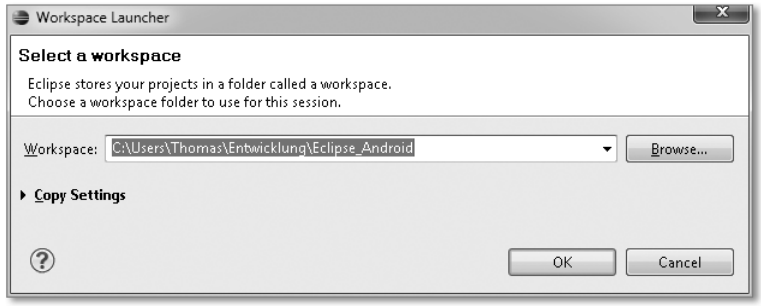


Abbildung B.1 Der Dialog »Workspace Launcher«

Nachdem Sie mit dem *Explorer* (Windows), *Finder* (Mac OS X) oder *Dolphin* bzw. *Nautilus* (beide Linux) die gewünschten Projekte in den Arbeitsbereich kopiert haben, können Sie diese importieren. Klicken Sie mit der rechten Maustaste in einen leeren Bereich der Sicht Package Explorer, und wählen Sie im daraufhin angezeigten Kontextmenü Refresh. Dieser Schritt ist nötig, damit Eclipse die in den Arbeitsbereich kopierten Projekte sofort findet.

Öffnen Sie erneut das Kontextmenü des Package Explorers. Klicken Sie diesmal auf Import. Sie sehen den in Abbildung B.2 gezeigten Assistenten zum Importieren von Projekten.

Markieren Sie Existing Projects into Workspace, und wechseln Sie anschließend mit Next auf die zweite Seite. Diese sehen Sie in Abbildung B.3.

Abbildung B.2 Assistent zum Importieren von Projekten

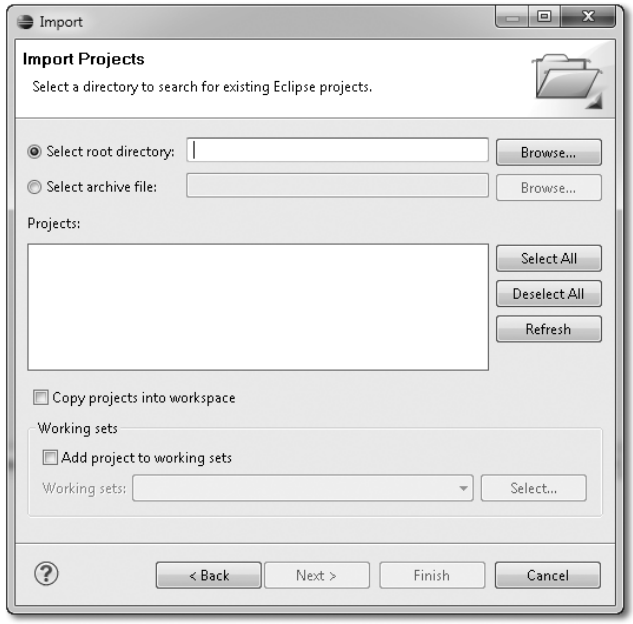


Abbildung B.3 Zweite Seite des Import-Assistenten

Stellen Sie sicher, dass SELECT ROOT DIRECTORY ausgewählt ist. Klicken Sie anschließend auf BROWSE..., um das Basisverzeichnis der zu importierenden Projekte auszuwählen. Da wir diese schon in den Eclipse-Arbeitsbereich kopiert haben, können Sie

den in Abbildung B.4 gezeigten Dialog mit OK schließen, ohne das voreingestellte Verzeichnis wechseln zu müssen.

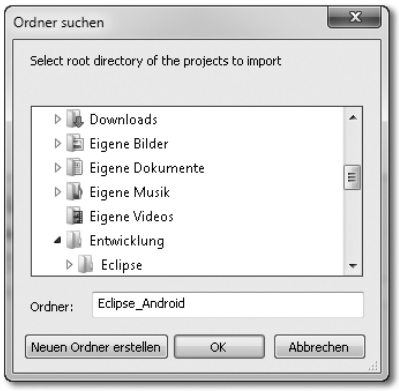


Abbildung B.4 Basisverzeichnis auswählen

Die zu importierenden Projekte können Sie unter PROJECTS mit einem Häkchen auswählen (siehe Abbildung B.5). SELECT ALL markiert alle Einträge. DESELECT ALL entfernt alle Häkchen. Klicken Sie auf FINISH, um Ihre Einstellungen zu übernehmen und den Assistenten abzuschließen. Die importierten Projekte erscheinen in der Sicht PACKAGE EXPLORER.

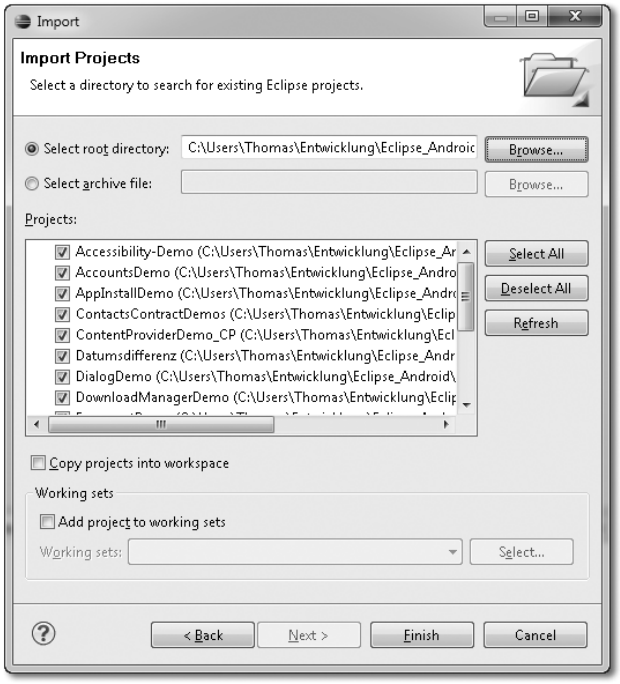


Abbildung B.5 Die zu importierenden Projekte auswählen

Sollten Fehler angezeigt werden, können Sie diese beheben, indem Sie alle Projekte neu bauen lassen. Stellen Sie hierfür sicher, dass der Eintrag BUILD AUTOMATICALLY des Menüs PROJECT mit einem Häkchen versehen ist. Mit PROJECT • CLEAN öffnen Sie den in Abbildung B.6 gezeigten Dialog CLEAN. Markieren Sie CLEAN ALL PROJECTS, und klicken Sie anschließend auf OK. Markieren Sie schließlich alle Projekte im *Package Explorer*, öffnen Sie dessen Kontextmenü, und klicken Sie auf REFRESH.

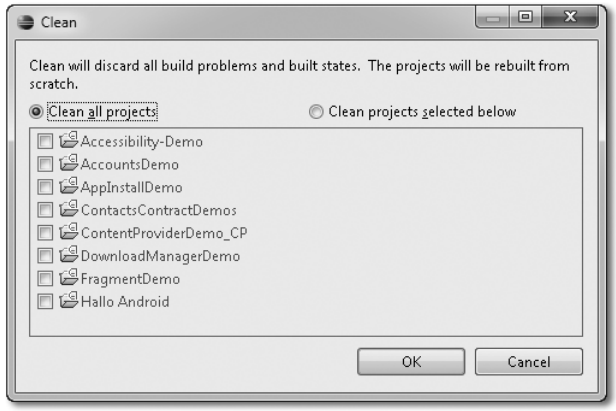


Abbildung B.6 Der Dialog »Clean«

Video-Training

Als besondere Beigabe finden Sie auf der Buch-DVD ein Kapitel des Video-Trainings »Apps entwickeln für Android 4« von Mike Bach, erschienen bei Galileo Press. (Mehr Informationen zu diesem Video-Training erhalten Sie auf *www.galileocomputing.de/2955*.) In diesem Kapitel wird gezeigt, wie Sie eine Notizzettel-App erstellen. Schon während Sie dem Trainer zuschauen, können Sie diese App nachbauen.

Um das Video-Training zu starten, legen Sie bitte die DVD-ROM in das DVD-Laufwerk Ihres Rechners ein. Der Kurs startet automatisch nach wenigen Augenblicken. Sollte das Training auf Ihrem PC nicht von alleine starten – beispielsweise weil in Ihrem System die Autoplay-Funktion ausgeschaltet ist –, so können Sie es auch selbst starten, indem Sie im Windows-Explorer im Verzeichnis »Video-Training« die Anwendungs-Datei »start.exe« per Doppelklick aufrufen.

Am *Mac* starten Sie das Video-Training mit der Datei »start.app«. Sollten Sie Probleme mit der Leistung Ihres Rechners feststellen, können Sie alternativ die Datei »start.html« aufrufen. Unter *Linux* rufen Sie bitte die Datei »Start_Linux.html« auf.

Bitte vergessen Sie nicht, die Lautsprecher zu aktivieren oder gegebenenfalls die Lautstärke zu erhöhen. Die erforderliche Bildschirmauflösung beträgt mindestens 1024 × 768 Pixel.

Wählen Sie im Hauptmenü per Mausklick ein Kapitel aus. Das jeweils ausgewählte Kapitel ist markiert. Bewegen Sie nun die Maus im rechten Feld des Hauptmenüs auf das Video, mit dem Sie starten wollen. Mit einem Klick rufen Sie das ausgewählte Video auf, und das Training beginnt! Aus dem laufenden Videokurs heraus können Sie nach Belieben mit einem Klick auf den Titel des Videos die Schnellnavigation aufrufen und jedes Kapitel und jedes dazugehörige Video auswählen.

Folgende Video-Lektionen können Sie sich ansehen:

1 Eine Notizzettel-App programmieren [02:09 Std.]

1.1 Einleitung [00:19 Min.]

1.2 Das Datenmodell der Listen [13:06 Min.]

1.3 Der Content-Provider [07:11 Min.]

1.4 Datenbank-Funktionen implementieren [10:38 Min.]

1.5 Beispieldaten einfügen und anzeigen [11:42 Min.]

1.6 Ein Kontextmenü hinzufügen [12:31 Min.]

1.7 Die Oberfläche der Listenbearbeitung [13:14 Min.]

1.8 Die Select-Funktionen vervollständigen [10:43 Min.]

1.9 Listen und Einträge anlegen [19:25 Min.]

1.10 Listen und Einträge löschen [09:24 Min.]

1.11 Listeneinträge aktualisieren [11:28 Min.]

1.12 Die Listen-App für Tablets [09:11 Min.]

Anhang C
Häufig benötigte Code-Bausteine

C.1 Manifestdatei

Rumpf

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
      xmlns:android="http://schemas.android.com/apk/res/android"
      package="paket.name.der.app"
      android:versionCode="1"
      android:versionName="1.0">
   <!-- mindestens benötigte SDK-Version -->
   <uses-sdk android:minSdkVersion="9" />
   <!-- angeforderte Berechtigungen -->
   <uses-permission android:name="..." />
   <application
      android:icon="@drawable/icon"
      android:label="@string/app_name">
   <activity
      android:name=".Name_der_Activity_Klasse"
      android:label="@string/app_name">
         <!-- Hauptaktivität kennzeichnen -->
         <intent-filter>
            <action
               android:name="android.intent.action.MAIN" />
            <category
               android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Broadcast Receiver

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

Service

```
<service
   android:label="@string/..."
   android:name=".Name_der_Service_Klasse"
   <!-- ggf. Berechtigungen anfordern -->
   android:permission="...">
   <intent-filter>
      <action
         android:name="..." />
   </intent-filter>
   <!-- ggf. Metadaten setzen -->
   <meta-data
      android:name="..."
      android:resource="@xml/..." />
</service>
```

Unterschiedliche Bildschirme unterstützen

```
<supports-screens
   android:smallScreens="..."
   android:normalScreens="..."
   android:largeScreens="..."
   android:xlargeScreens="..." />
```

Content Provider

```
<provider
   android:name=".Name_der_Content_Provider_Klasse"
   android:authorities="..." />
```

Benötigte Features

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

C.2 Berechtigungen

C.2.1 Hardware, Telefonie und Netzwerk

Anrufe tätigen

```
android.permission.CALL_PHONE
```

Telefonstatus auslesen

```
android.permission.READ_PHONE_STATE
```

Netzwerkstatus abrufen

```
android.permission.ACCESS_NETWORK_STATE
```

Zugriff auf Location Provider

```
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION
```

Auf externes Medium schreiben

```
android.permission.WRITE_EXTERNAL_STORAGE
```

Wallpaper setzen

```
android.permission.SET_WALLPAPER
```

Live Wallpaper setzen

```
android.permission.BIND_WALLPAPER
```

C.2.2 Internet

Zugriff auf Internet

```
android.permission.INTERNET
```

Auf Browser-Bookmarks und -Historie zugreifen

```
com.android.browser.permission.READ_HISTORY_BOOKMARKS
com.android.browser.permission.WRITE_HISTORY_BOOKMARKS
```

Auf Online-Konten zugreifen

```
android.permission.GET_ACCOUNTS
android.permission.USE_CREDENTIALS
```

C.2.3 Audio und Video

Audio-Einstellungen ändern

```
android.permission.MODIFY_AUDIO_SETTINGS
```

Audio aufnehmen

```
android.permission.RECORD_AUDIO
```

Auf Kamera zugreifen

```
android.permission.CAMERA
```

C.2.4 Kontakte und Kalender

Auf Kalender zugreifen

```
android.permission.READ_CALENDAR
android.permission.WRITE_CALENDAR
```

Alarm setzen

```
com.android.alarm.permission.SET_ALARM
```

Auf Anrufhistorie und Kontakte zugreifen

```
android.permission.READ_CONTACTS
android.permission.WRITE_CONTACTS
```

Auf Benutzerwörterbuch zugreifen

```
android.permission.READ_USER_DICTIONARY
android.permission.WRITE_USER_DICTIONARY
```

Index

.3gp ... 336
.wav ... 358
@drawable ... 91, 225
@string ... 89, 225
_ID ... 404

A

ACCESS_COARSE_LOCATION ... 241
ACCESS_FINE_LOCATION ... 241, 244
ACCESS_NETWORK_STATE ... 190
Account ... 413
AccountManager ... 409, 412, 413, 414
AccountManagerCallback ... 414
action ... 103
Action Bar ... 24, 150, 153
action item ... 155, 156
action view ... 156
ACTION_AUDIO_BECOMING_NOISY ... 366
ACTION_BOOT_COMPLETED ... 208
ACTION_CALL ... 186
ACTION_CHECK_TTS_DATA ... 353
ACTION_CHOOSER ... 210
ACTION_DIAL ... 186
ACTION_EDIT ... 418
ACTION_IMAGE_CAPTURE ... 370, 372
ACTION_INSERT ... 104
ACTION_MEDIA_BUTTON ... 363, 365
ACTION_RECOGNIZE_SPEECH ... 361
ACTION_SEND ... 377, 380
ACTION_SET_ALARM ... 419, 420, 421
ACTION_SET_WALLPAPER ... 209
ACTION_VIEW ... 376
ACTION_WEB_SEARCH ... 103
ActionBarDemo1 ... 153
Activity ... 41, 43, 54, 72, 75, 87, 89, 93, 103, 110, 137, 144, 148, 160, 211, 227, 369, 372, 382, 412, 417, 423
Activity Manager ... 29
Adapter ... 68, 337
adb ... 84
add() ... 337
addAccount() ... 413
addCallback() ... 382, 384
addHeader() ... 416
addPreferencesFromResource() ... 142, 222
addTab() ... 141
addView() ... 124
ADT ... 29, 39, 41, 43, 50, 54, 80, 136
AIDL ... 177
AlarmClock ... 419
AlarmManager ... 422
AlertDialog ... 146, 148
AlertDialog.Builder ... 145
Alternative Ressourcen ... 130
AnalogClock ... 120
Android Backup Service ... 310
Android Backup Service Key ... 312
Android Debug Bridge ... 231, 271
Android Development Tools → ADT
Android Device Chooser ... 46
Android Inc ... 22
Android Interface Definition Language ... 177
Android Runtime ... 28
Android SDK ... 29, 30
Android SDK Manager ... 36, 243, 409
Android SDK Platform Tools ... 31
Android support package ... 109
Android Virtual Device ... 32, 316
Android Virtual Device → AVD
android.appwidget.provider ... 197
ANDROID_ID ... 189
AndroidManifest.xml ... 43, 50, 88
ANR ... 166
Apache Harmony → Harmony
apiKey ... 244, 247
API-Schlüssel ... 243
apk ... 80
APP_ID ... 292
application ... 43, 76, 89, 109, 194
Application Framework ... 28
Application Info ... 40
Application not responding ... 166
Application Package ... 80
APPWIDGET_UPDATE ... 196
AppWidgetManager ... 202
AppWidgetProvider ... 196, 199, 201, 206
Arbeitsbereich ... 39, 63
ArrayAdapter ... 68, 337
ArrayList ... 71
Aspect Ratio ... 135
assets ... 41, 43
attachAuxEffect() ... 349
AudioEffect ... 347

AudioManager ... 363
Auflösung ... 135
Auth Token ... 413
Auth Token-Typ ... 413
Authority ... 302
AVD ... 32, 44, 241, 316
AVD Manager ... 32

B

Back Stack ... 88, 100
background ... 126
Backup Agent ... 309, 312
Backup Helper ... 316
Backup Manager ... 309, 310, 315, 330
Backup Service Key ... 311
Backup Transport ... 309, 310
BackupDemo1 ... 319, 320
BackupDemo2 ... 319
BackupDemo3 ... 323
BassBoost ... 347
Benutzeroberfläche ... 66
Bildschirmgröße ... 135
bin ... 29
BIND_WALLPAPER ... 214
Binder ... 174, 175
bindService() ... 176, 179
Bitmap ... 209, 374, 379
BOOKMARK ... 295, 296
BookmarkColumns ... 295
Bornstein, Dan ... 27
Broadcast Receiver ... 106
BroadcastReceiver ... 196, 365
Browser ... 295, 297
Button ... 53, 118, 345
Bytecode ... 26

C

Cache Flush ... 164
Call Log ... 170, 185, 190, 290
CALL_PHONE ... 186
CALL_STATE_IDLE ... 187
CALL_STATE_OFFHOOK ... 187
CALL_STATE_RINGING ... 187
Callback ... 393
CallLog ... 191
CAMERA ... 385, 390
Camera ... 384
CameraInfo ... 386
Canvas ... 209, 379
CATEGORY_BROWSABLE ... 103
center ... 126
CHECK_VOICE_DATA_PASS ... 353
CheckBox ... 122
CheckBoxPreference ... 143
Codd, E. F. ... 269
ColorMatrix ... 379
com.google ... 413
CommonDataKinds ... 400
configure ... 206
ConnectivityManager ... 190
connectSimulator() ... 237
CONTACT_ID ... 400, 402
Contacts ... 398, 399
ContactsContract ... 397, 399, 400
Content Provider ... 29, 290, 404
Content Resolver ... 408, 423
content:// ... 302
CONTENT_ITEM_TYPE ... 400, 402
CONTENT_URI ... 292, 293, 298, 303, 397, 400
ContentObserver ... 170, 194, 197, 199
ContentProvider ... 302, 303
ContentResolver ... 192, 193, 289, 290, 291, 293, 294, 295, 297, 298, 300, 398
ContentValues ... 292, 294, 295, 298, 404
Context ... 144, 186, 192, 201, 208, 227, 290
convert() ... 242
convertView ... 119
Core Libraries ... 28
Create Activity ... 41, 94
create() ... 145
createChooser() ... 380
createNewFile() ... 343
Cupcake ... 23, 41, 195
currentThread() ... 165
Cursor ... 289, 294, 300
CursorAdapter ... 300
Custom Locale ... 91

D

Dalvik ... 25, 27, 174
Dalvik Debug Monitor Server → DDMS
Dalvik Executable ... 26
Danger Inc. ... 23
Data ... 398
Date ... 336
Datenbankmanagementsystem ... 270, 272
Datenfeld ... 269
Datensatz ... 269
DatePicker ... 130, 145

DatePickerDialog ... 145, 146
DDMS ... 188, 241, 247
DebugDemo ... 72
debuggable ... 76
delete() ... 290, 293, 297, 300, 306, 373
Density-independent Pixel → dp
Developer Console ... 80, 82
Dialog ... 148
DialogFragment ... 111
Display Saved Filters View ... 73
Donut ... 23, 134, 350
dp ... 137, 203
drawable-hdpi ... 91, 136
drawable-ldpi ... 91, 136
drawable-mdpi ... 136
drawBitmap() ... 379
dx ... 27

E

Eclair ... 24
Eclipse ... 29
EditText ... 53, 59, 118
EditTextPreference ... 143, 144
Eingabeaufforderung ... 33, 271, 317, 318
Emulator ... 33, 46
Emulator Control ... 189, 193, 241
enabled ... 108
Entity ... 330
Environment ... 339
Event ... 400, 402
Event Dispatching Thread ... 167
exported ... 108, 171, 175, 224
EXTRA_HOUR ... 419
EXTRA_LANGUAGE ... 362
EXTRA_MESSAGE ... 419
EXTRA_MINUTES ... 419
EXTRA_SKIP_UI ... 419

F

Fibonacci-Folge ... 76, 161
File ... 336, 343
File Explorer ... 207
FilenameFilter ... 338
fill_parent ... 53
findViewById() ... 71, 119
finish() ... 100, 353
FORMAT_MINUTES ... 242
Fragment ... 24, 109, 111
FrameLayout ... 118, 121, 124
FREQUENCY ... 292
Froyo ... 24, 263

G

G1 ... 15, 21, 23, 130, 136
gen ... 43
GeoPoint ... 245, 247
GET_ACCOUNTS ... 416
getAccountsByType() ... 412
getAction() ... 211
getAllNetworkInfo() ... 190
getAllProviders() ... 238
getAppWidgetIds() ... 202
getAudioSessionId() ... 347
getAuthToken() ... 413, 414
getBinder() ... 177
getBitmap() ... 374, 377
getCameraInfo() ... 386
getColumnIndex() ... 292
getContentResolver() ... 192, 290, 294, 295
getCount() ... 71
getData() ... 375
getDefaultSensor() ... 228
getDesiredMinimumHeight() ... 208
getDesiredMinimumWidth() ... 208
getDeviceId() ... 189
getExternalStorageState() ... 339
getHeight() ... 119
getId() ... 349
getInstance() ... 202, 208
getIntent() ... 421
getIntExtra() ... 421
getItem() ... 71
getKeyCode() ... 366
getLastKnownLocation() ... 239, 242
getListView() ... 67, 139
getMainLooper() ... 358
getMaximumRange() ... 228
getMeasuredHeight() ... 119
getMeasuredWidth() ... 119
getName() ... 165
getNetworkInfo() ... 190
getPaddingBottom() ... 120
getPaddingLeft() ... 120
getPaddingRight() ... 120
getPaddingTop() ... 120
getPower() ... 228
getProvider() ... 238
getResolution() ... 228
getResources() ... 93

getResult() ... 414
getRoundedStrength() ... 347
getSensorList() ... 228
getService() ... 175
getStrengthSupported() ... 347
getString() ... 58, 93, 144
getStringArrayListExtra() ... 362
getStringExtra() ... 421
getSupportedPreviewSizes() ... 384
getSystemService() ... 187, 227, 237
getType() ... 302, 304
getView() ... 71, 119
getWidth() ... 119
getWritableDatabase() ... 304, 305
Gingerbread ... 24, 41, 344, 386, 419
Global Positioning System → GPS
Google APIs ... 409
Google APIs Console ... 414
Google Maps ... 243
Google Music ... 25
Google Play .. 25, 32, 44, 62, 71, 79, 81, 91, 230, 243, 385
Google-APIs ... 243
Google-Konto ... 408
Google-Maps-API ... 244
GPS ... 238
gravity ... 124, 127

H

handleMessage() ... 177, 180
Handler ... 169, 180
Harmony ... 28, 160
HashMap ... 356
Hashtable ... 65
hierarchyviewer ... 128
Hiptop ... 23
Honeycomb ... 24, 109, 148, 150, 153, 419
HttpGet ... 416

I

IBinder ... 170, 174
Ice Cream Sandwich ... 24, 109, 417
icon ... 79, 91
Icon Design ... 91
Images ... 374
ImageView ... 374
IN_VISIBLE_GROUP ... 399
Input Method Framework ... 23
inputType ... 59
insert() ... 290, 292, 295, 298, 302, 304, 404
instance state ... 95
Intent .. 102, 140, 173, 209, 363, 365, 376, 377, 380, 418
explizites ... 102, 104
implizites ... 102, 104
INTENT_ACTION_STILL_IMAGE_CAMERA ... 369, 370
INTENT_ACTION_VIDEO_CAMERA ... 369, 370
Intent-Filter ... 102, 178, 379
intent-filter ... 89, 94, 103
INTERNET ... 237, 244, 416
interrupt() ... 162
InterruptedException ... 162
invalidateOptionsMenu() ... 150
isAlive() ... 161
isAvailable() ... 190
isChecked() ... 123
isLanguageAvailable() ... 356
isLongPress() ... 366
isMusicActive() ... 364
isProviderEnabled() ... 238
isRoaming() ... 190

J

Java Development Kit ... 29
Java Virtual Machine ... 26
javac ... 29
JavaScript Object Notation ... 324
JDK ... 29
JSON ... 324
Just-in-time-Compiler ... 26
JVM ... 26

K

KEY_AUTHTOKEN ... 414
KeyEvent ... 365
Keystore ... 80
keytool ... 80, 243
Kindle Fire ... 78
KontakteDemo3 ... 407
Kontextmenü ... 151, 284

L

LABEL ... 401
label ... 89
LANGUAGE_MODEL_FREE_FORM ... 362
LANGUAGE_MODEL_WEB_SEARCH ... 362

Layout 51
layout 52, 117
layout_above 130
layout_below 66
layout_gravity 126
layout_height 53, 118
layout_toLeftOf 130
layout_toRightOf 66
layout_weight 128
layout_width 53, 118
Layoutdatei 52
LayoutInflator 71
LinearLayout 53, 67, 118, 124, 128, 381
Linux 26, 159
ListActivity 67, 94, 139
ListAdapter 68, 139
listen() 187
LISTEN_CALL_STATE 187
LISTEN_NONE 187
listFiles() 338
ListFragment 111
ListPreference 225
ListView 66
Live Wallpaper 24
Location 242, 247
LOCATION_SERVICE 237
LocationListener 239, 241, 245
LocationManager 237
LocationProvider 238
Log 73
LogCat 72, 73, 92, 161, 162, 164, 168, 169, 290, 294, 318, 331, 363, 366, 397, 399, 402, 419, 423
LogView 77
Looper 358

M

Magic 130, 136
Magic Cap 23
MAIN 103
Mainthread 165
manifest 43, 387
Manifestdatei 379, 419, 423
MapActivity 243
MapController 245
MapView 242, 244, 247
Media 374
MediaPlayer 341, 342, 345, 348
MediaRecorder 342, 343, 391
MediaStore 369, 370, 372, 374
MenuDemo 149, 150
Message 178
Messenger 177, 180
MIME-Typ 379
MIMETYPE 402
Miner, Richard 23
minHeight 206
minSdkVersion 44, 89, 109, 136
minWidth 206
MODIFY_AUDIO_SETTINGS 347
moveToNext() 292
Multitasking 159
präemptives 159

N

name 49, 103
Notification Manager 29
NullPointerException 292

O

obtain() 179
OHA 22
onActivityCreated() 114
onActivityResult() 105, 106, 353, 361, 362, 372, 373, 375
onAttach() 114
onBind() 170, 174
onCallStateChanged() 187
onChange() 170, 194
onClick() 167, 168, 169, 338, 369
OnClickListener 58, 122, 338, 369, 387
OnCompletionListener 341, 345
onContextItemSelected() 151, 300
onCreate() 57, 59, 72, 75, 95, 96, 97, 99, 114, 140, 142, 164, 170, 187, 193, 194, 228, 237, 245, 295, 300, 302, 304, 338, 345, 351, 354, 361, 363, 369, 377, 382, 387, 391, 409, 412, 418, 421, 423
onCreateContextMenu() 151
onCreateDialog() 146, 147, 148
onCreateEngine() 216
onCreateOptionsMenu() 149, 150
onCreateView() 111, 112
OnDateSetListener 145
onDestroy() 114, 170, 194, 199, 228, 345, 351, 352
onDestroyView() 112
onDetach() 114
onDisabled() 199, 200
onDraw() 118
onEnabled() 199, 200
onInfo() 395

OnInfoListener ... 395
onInit() ... 354, 357
OnInitListener ... 354
OnItemClickListener ... 68
onLocationChanged() ... 239, 242
onOptionsItemSelected() ... 149
onPause() 97, 100, 114, 228, 338, 363, 364, 382, 384
onPictureTaken() ... 390
onPrepareDialog() ... 147
onPrepareOptionsMenu() ... 150
onReceive() ... 107, 200, 365
onRestart() ... 100
onResume() ... 99, 363, 364, 382, 387
onSaveInstanceState() ... 97
onSensorChanged() ... 233
onServiceConnected() ... 176
onShutter() ... 390
onStart() ... 97, 99, 164, 176, 179, 228
onStartCommand() ... 173
onStop() ... 114
onUpdate() ... 201, 204, 206
onUtteranceCompleted() ... 357
Open Handset Alliance ... 22
Open Source ... 22
open() ... 384
Optionsmenü ... 87, 148
orientation ... 127
Orientierungswechsel ... 95
OutOfMemoryError ... 377

P

package ... 89, 102
Package Explorer ... 43, 80, 132
Parameters ... 390
Parcelable ... 96
PATH ... 29, 30, 84
PendingIntent ... 422
permission ... 109, 171
PhoneStateListener ... 191
Pico ... 350
PictureCallback ... 388
Pixeldichte ... 135
Plattform ... 25
Play Store ... 25
post() ... 169, 358
postDelayed() ... 216, 225
Präemptives Multitasking ... 159
PreferenceActivity ... 222
PreferenceCategory ... 143
prepare() ... 341, 343
PresetReverb ... 349
process ... 109, 171, 172
put() ... 292
putExtra() ... 105, 362, 372, 418

Q

query() ... 289, 290, 292, 294, 302, 305, 400, 423

R

R ... 43, 50
R.string ... 63
RatingBar ... 120
RawContacts ... 398, 404
READ_CALENDAR ... 423
READ_CONTACTS ... 191, 398
READ_HISTORY_BOOKMARKS ... 296
READ_PHONE_STATE ... 188, 189
READ_USER_DICTIONARY ... 293
RECEIVE_BOOT_COMPLETED ... 108
receiver ... 108, 196, 200
RecognizerIntent ... 361
RECORD_AUDIO ... 390
recycle() ... 377
registerForContextMenu() ... 151
registerListener() ... 228
registerMediaButtonEventReceiver() ... 364
registerReceiver() ... 108
Relationales Datenbanksystem ... 269
RelativeLayout ... 66, 129
release() ... 345, 347, 384
Remote Procedure Call ... 174
RemoteView ... 205
removeCallback() ... 382
removeCallbacks() ... 219
replyTo ... 178, 180
Request Code ... 105
requestCode ... 373
requestLocationUpdates() ... 239
res ... 41, 52, 90, 117
resource ... 197
Resource Manager ... 29
resources ... 220
RESULT_OK ... 373
resultCode ... 373
Rubin, Andy ... 23
Run Configuration ... 44
run() ... 161
Runnable ... 161, 169, 216

S

saveBookmark() ... 297
Schlüssel ... 80
ScrollView ... 135
SDK Manager ... 30
Sears, Nick ... 23
Secondary Market ... 80, 84
Seitenverhältnis ... 135
Select Build Target ... 35, 40
Select Sample ... 35
send() ... 179
Sensor ... 227
Sensor Simulator ... 230
SENSOR_SERVICE ... 227
SensorEventListener ... 228, 233
SensorManager ... 227, 230
Service ... 170, 177, 193, 227
service ... 171, 172, 215
ServiceConnection ... 176
set() ... 422
SET_ALARM ... 419
SET_WALLPAPER ... 208, 211
setAudioEncoder() ... 342
setAudioSource() ... 342
setAuxEffectSendLevel() ... 349
setBitmap() ... 209, 211
setBuiltInZoomControls() ... 247
setCenter() ... 245
setChecked() ... 123
setContentView() ... 95, 118, 134, 137, 244, 354
setDataSource() ... 341
setEnabled() ... 347
setImageBitmap() ... 374
setMaxDuration() ... 394
setMaxFileSize() ... 395
setOnClickListener() ... 123, 387
setOnClickPendingIntent() ... 204
setOnItemClickListener() ... 139
setOnUtteranceCompletedListener() ... 357
setOutputFormat() ... 342
setPadding() ... 120
setPreviewDisplay() ... 394
setPreviewSize() ... 384
setRequestedOrientation() ... 372
setResource() ... 208
setRingerMode() ... 364
setSaturation() ... 379
setStrength() ... 347
setTag() ... 71
setText() ... 112
setTextViewText() ... 205
Settings.Secure ... 189
settingsActivity ... 224
setType() ... 382
Shell ... 33, 271, 317, 318
showDialog() ... 145, 148
shutdown() ... 352
ShutterCallback ... 388
Simple API Access ... 414
SimpleCursorAdapter ... 139
SIP ... 24
sleep() ... 162
Software Development Kit → Android SDK
Source attachment ... 36
Source Attachment Configuration ... 37
Sources for Android SDK ... 36
speak() ... 356
SQL ... 270
SQLite ... 269, 270, 291
SQLiteDatabase ... 293
SQLiteOpenHelper ... 298
SQLiteQueryBuilder ... 305
src ... 43
Stacktrace ... 75
start() ... 341, 343, 347
START_DATE ... 401
START_NOT_STICKY ... 173
START_REDELIVER_INTENT ... 173
START_STICKY ... 173
startActivity() ... 104, 186, 209, 376, 380
startActivityForResult() ... 105, 353, 362, 370, 373
Starter Package ... 30
startForeground() ... 199
startPreview() ... 382
startService() ... 171, 172, 173, 194, 197
stop() ... 162, 343
stopSelf() ... 172
stopSelfResult() ... 172
stopService() ... 172, 197
string ... 49
strings.xml ... 43, 49, 65, 89
Structured Query Language ... 270
SuperNotCalledException ... 95
supports-screens ... 136
Surface Manager ... 27
surfaceCreated() ... 384
SurfaceHolder ... 382, 393
SurfaceHolder.Callback ... 382
SurfaceView ... 381, 383, 391, 393
Swing ... 167
synchronized ... 164

synthesizeToFile() ... 358
System Bar ... 24, 88, 151

T

TabActivity ... 140
TabContentFactory ... 141
TabHost ... 140, 141
TabSpec ... 140, 141
takePicture() ... 388, 390
targetSdkVersion ... 109, 135
TELEPHONY_SERVICE ... 187
TelephonyManager ... 189
Telnet ... 188, 241
TextToSpeech ... 351, 356
TextToSpeech.Engine ... 353
TextView ... 53, 111, 337
Thread ... 160, 161, 162, 165
thumbnail ... 225
Tierkreiszeichen ... 65
TITLE ... 296
Toast ... 275, 299
toString() ... 336
Treiber ... 78
Tupel ... 269
TYPE ... 401
TYPE_ALL ... 228
TYPE_WIFI ... 190

U

UI-Thread ... 165, 167, 168
Umgebungsvariable ... 29, 33, 84
Unbekannte Herkunft ... 84
unbindService() ... 176
unregisterListener() ... 228
unregisterMediaButtonEventReceiver() ... 364
update() ... 289, 290, 300, 302, 306
updatePeriodMillis ... 206
URI ... 303, 304, 306
UriMatcher ... 303, 304
URL ... 295, 297
USB-Debugging ... 78
USB-Port ... 78
USE_CREDENTIALS ... 416
User Credentials ... 413
UserDictionary ... 293
UserDictionaryDemo ... 290
uses-feature ... 222, 230
uses-library ... 244
uses-permission ... 108
uses-sdk ... 43, 89

V

values ... 43, 49, 50, 90
values-en ... 50
values-fr ... 50
versionCode ... 89
versionName ... 89
View ... 28, 51, 118
ViewGroup ... 51, 54, 124
ViewHolder ... 71
Virtualizer ... 348, 349
Virtuelle Maschine Dalvik ... 26
VISITS ... 295, 297
VoIP ... 24
volatile ... 164

W

wallpaper ... 215, 225
WallpaperManager ... 208
WallpaperService ... 214, 215
weightSum ... 128
White, Chris ... 23
Widget Preview ... 206
Widgets ... 24
Wireframe ... 51, 66
withAppendedPath() ... 300
WORD ... 292
Words ... 292
Workspace → Arbeitsbereich
wrap_content ... 53
WRITE_CALENDAR ... 423
WRITE_CONTACTS ... 193, 404
WRITE_EXTERNAL_STORAGE ... 391
WRITE_HISTORY_BOOKMARKS ... 296
WRITE_USER_DICTIONARY ... 293

Y

yield() ... 161

Z

Zurück-Taste ... 88