

Joseph Annuzzi, Jr.
Lauren Darcey
Shane Conder



Fourth Edition

Advanced Android™ Application Development

Developer's Library



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Praise for *Advanced Android™ Application Development, Fourth Edition*

“This new edition of *Advanced Android™ Application Development* updates the definitive reference for Android developers, covering all major revisions of Android, including Android L. Whether you’re just getting started, or need to brush up on the latest features of Android, this should be the first book you reach for.”

—Ray Rischpater, senior software engineer, Microsoft

“This is the most comprehensive reference for programming Android. I still turn to it when I need to learn about a topic I am not familiar with.”

—Douglas Jones, senior software engineer, Fullpower Technologies

“The problem with many Android development titles is that they either assume the developer is completely new to development or is already an expert. *Advanced Android™ Application Development, Fourth Edition*, cuts the fluff and gets to the need to know of modern Android development.”

—Phil Dutson, solution architect for mobile and UX, ICON Health & Fitness

“*Advanced Android™ Application Development, Fourth Edition*, is an excellent guide for software developers, quality assurance personnel, and project managers who want to learn to plan, develop, and manage professional Android applications. The book explains several advanced Android topics through step-by-step running examples. The authors have done a great job explaining various Android APIs for threading, networking, location-based services, hardware sensors, animation, graphics, and more. This book is a classic investment.”

—B.M. Harvani, author, *The Android™ Tablet Developer’s Cookbook*

This page intentionally left blank

Advanced Android™ Application Development

Fourth Edition

Developer's Library Series



Visit developers-library.com for a complete list of available products

The **Developer's Library Series** from Addison-Wesley provides practicing programmers with unique, high-quality references and tutorials on the latest programming languages and technologies they use in their daily work. All books in the Developer's Library are written by expert technology practitioners who are exceptionally skilled at organizing and presenting information in a way that's useful for other programmers.

Developer's Library books cover a wide range of topics, from open-source programming languages and databases, Linux programming, Microsoft, and Java, to Web development, social networking platforms, Mac/iPhone programming, and Android programming.

PEARSON

Advanced Android™ Application Development

Fourth Edition

Joseph Annuzzi, Jr.
Lauren Darcey
Shane Conder

◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Revision of: Android wireless application development. Volume II, Advanced topics. ©2012.

Includes bibliographical references and index.

Summary: "This book—a renamed new edition of Android Wireless Application Development, Volume II—is the definitive guide to advanced commercial-grade Android development, updated for the latest Android SDK. The book serves as a reference for the Android API."— Provided by publisher.

ISBN 978-0-13-389238-3 (pbk. : alk. paper)

1. Application software—Development. 2. Android (Electronic resource) 3. Mobile computing. 4. Wireless communication systems. I. Darcey, Lauren, 1977- author. II. Conder, Shane, 1975- author. III. Title.

QA76.76.A65A55 2015

004.167—dc23

2014033049

Copyright © 2015 Joseph Annuzzi, Jr., Lauren Darcey, and Shane Conder

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

Some figures that appear in this book have been reproduced from or are modifications based on work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 3.0 Attribution License. (<https://creativecommons.org/licenses/by/3.0/>).

Some figures that appear in this book have been reproduced from or are modifications based on work created and shared by Google and used according to terms described in the Creative Commons Attribution 3.0 License. See <https://developers.google.com/site-policies>.

Screenshots of Google products follow these guidelines:

<http://www.google.com/permissions/using-product-graphics.html>

The following are registered trademarks of Google:

Android™, Chrome™, Google Play™, Google Wallet™, Nexus™, Google Analytics™, Dalvik™, Daydream™, Google Maps™, Google TV™, Google and the Google logo are registered trademarks of Google Inc.

ISBN-13: 978-0-13-389238-3

ISBN-10: 0-13-389238-7

Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.

First printing, November 2014

Editor-in-Chief

Mark L. Taub

Executive Editor

Laura Lewin

Development Editor

Songlin Qiu

Managing Editor

John Fuller

Full-Service

Production Manager

Julie B. Nahil

Project Manager

Thistle Hill Publishing Services

Copy Editor

Barbara Wood

Indexer

Jack Lewis

Proofreader

Melissa Panagos

Technical Reviews

Douglas Jones
Raymond Rischpater
Valerie Shipbaugh

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Shepherd, Inc.



This book is dedicated to Cleopatra (Cleo).

—Joseph Annuzzi, Jr.

This book is dedicated to ESC.

—Lauren Darcey and Shane Conder



This page intentionally left blank

Contents at a Glance

Contents xiii

Acknowledgments xxxiii

About the Authors xxxv

Introduction 1

I: Advanced Android Application Design Principles 9

- 1 Threading and Asynchronous Processing 11
- 2 Working with Services 19
- 3 Leveraging SQLite Application Databases 35
- 4 Building Android Content Providers 55
- 5 Broadcasting and Receiving Intents 67
- 6 Working with Notifications 77

II: Advanced Android User Interface Design Principles 95

- 7 Designing Powerful User Interfaces 97
- 8 Handling Advanced User Input 115
- 9 Designing Accessible Applications 139
- 10 Development Best Practices for Tablets, TVs, and Wearables 151

III: Leveraging Common Android APIs 161

- 11 Using Android Networking APIs 163
- 12 Using Android Web APIs 175
- 13 Using Android Multimedia APIs 191
- 14 Using Android Telephony APIs 211
- 15 Accessing Android's Hardware Sensors 225
- 16 Using Android's Optional Hardware APIs 235

IV: Leveraging Google APIs 251

- 17** Using Location and Map APIs **253**
- 18** Working with Google Cloud Messaging **271**
- 19** An Overview of In-App Billing APIs for Android **277**
- 20** Enabling Application Statistics with Google Analytics **283**
- 21** An Overview of Google Play Game Services **295**

V: Drawing, Animations, and Graphics Programming with Android 303

- 22** Developing Android 2D Graphics Applications **305**
- 23** Working with Animation **329**
- 24** Developing Android 3D Graphics Applications **345**
- 25** Using the Android NDK **377**

VI: Maximizing Android's Unique Features 389

- 26** Extending Android Application Reach **391**
- 27** Enabling Application Search **415**
- 28** Managing User Accounts and Synchronizing User Data **427**

VII: Advanced Topics in Application Publication and Distribution 437

- 29** Internationalizing Your Applications **439**
- 30** Protecting Applications from Software Piracy **449**

VIII: Preparing for Future Android Releases 457

- 31** Introducing the L Developer Preview **459**

IX: Appendixes 467**A Quick-Start Guide: Android Debug Bridge 469****B Quick-Start Guide: SQLite 485****C Java for Android Developers 499****D Quick-Start Guide: Android Studio 507****E Answers to Quiz Questions 519****Index 527**

This page intentionally left blank

Contents

Acknowledgments xxxiii

About the Authors xxxv

Introduction **1**

Who Should Read This Book? 1

How This Book Is Structured 1

Key Questions Answered in This Book 3

An Overview of Changes in This Edition 4

The Development Environment Used in This Book 5

Supplementary Materials Available 6

Where to Find More Information 6

Conventions Used in This Book 7

Contacting the Authors 8

I: Advanced Android Application Design Principles 9

1 Threading and Asynchronous Processing 11

The Importance of Processing Asynchronously 11

Working with the `AsyncTask` Class 12

Working with the `Thread` Class 15

Working with `Loaders` 16

Understanding `StrictMode` 17

Summary 17

Quiz Questions 17

Exercises 18

References and More Information 18

2 Working with Services 19

Determining When to Use Services 19

Understanding the `Service` Lifecycle 20

Creating a `Service` 20

Controlling a `Service` 25

Implementing a Remote Interface 26

Implementing a `Parcelable` Class 28

Using the `IntentService` Class 30

Summary 33

Quiz Questions 34

Exercises 34

References and More Information 34

3 Leveraging SQLite Application Databases 35

Storing Structured Data Using SQLite Databases 35

Creating a SQLite Database 36

Creating, Updating, and Deleting Database Records 38

Working with Transactions 40

Querying SQLite Databases 40

Working with Cursors 41

Executing Simple Queries 43

Executing More Complex Queries Using `SQLiteQueryBuilder` 44

Executing Raw Queries without Builders and Column Mapping 45

Closing and Deleting a SQLite Database 45

Deleting Tables and Other SQLite Objects 46

Closing a SQLite Database 46

Deleting a SQLite Database Instance Using the `ApplicationContext` 46

Designing Persistent Databases 46

Keeping Track of Database Field Names 47

Extending the `SQLiteOpenHelper` Class 47

Binding Data to the Application User Interface 48

Working with Database Data Like Any Other Data 49

Binding Data to Controls Using Data Adapters 50

Summary 53

Quiz Questions 54

Exercises 54

References and More Information 54

4 Building Android Content Providers 55

Acting as a Content Provider 55

Implementing a Content Provider Interface 55

Defining the Data URI 56

Defining Data Columns 56

Implementing Important Content Provider Methods	57
Updating the Manifest File	62
Enhancing Applications Using Content Providers	62
Summary	65
Quiz Questions	65
Exercises	65
References and More Information	65
5 Broadcasting and Receiving Intents	67
Sending Broadcasts	67
Sending Basic Broadcasts	68
Sending Ordered Broadcasts	68
Receiving Broadcasts	69
Registering to Receive Broadcasts	70
Handling Incoming Broadcasts from the System	71
Securing Application Broadcasts	73
Summary	74
Quiz Questions	74
Exercises	75
References and More Information	75
6 Working with Notifications	77
Notifying the User	77
A Word on Compatibility	78
Notifying with the Status Bar	78
Using the <code>NotificationManager</code> Service	79
Creating a Simple Text Notification with an Icon	79
Working with the Notification Queue	80
Updating Notifications	81
Clearing Notifications	82
Vibrating the Phone	84
Blinking the Lights	84
Making Noise	86
Customizing the Notification	86
Expandable and Contractible Notifications	88
Notification Priority	90
Introducing the Notification Listener	91

Designing Useful Notifications	91
Summary	92
Quiz Questions	92
Exercises	92
References and More Information	92

II: Advanced Android User Interface Design Principles 95

7 Designing Powerful User Interfaces 97	
Following Android User Interface Guidelines	97
Enabling Action Bars	98
Building Basic Action Bars	98
Customizing Your Action Bar	101
Handling Application Icon Clicks on the Action Bar	103
Working with Screens That Do Not Require Action Bars	104
Contextual Action Mode	105
Working with Styles	106
Building Simple Styles	106
Leveraging Style Inheritance	109
Working with Themes	111
Summary	113
Quiz Questions	113
Exercises	113
References and More Information	114
8 Handling Advanced User Input 115	
Working with Textual Input Methods	115
Working with Software Keyboards	115
Working with Text Prediction and User Dictionaries	118
Using the Clipboard Framework	118
Handling User Events	119
Listening for Touch Mode Changes	119
Listening for Events on the Entire Screen	120
Listening for Long Clicks	121
Listening for Focus Changes	122

Working with Gestures	123
Detecting User Motions within a View	123
Handling Common Single-Touch Gestures	124
Handling Common Multitouch Gestures	129
Making Gestures Look Natural	133
Using the Drag-and-Drop Framework	134
Handling Screen Orientation Changes	134
Summary	137
Quiz Questions	137
Exercises	137
References and More Information	137
9 Designing Accessible Applications	139
Exploring the Accessibility Framework	139
Leveraging Speech Recognition Services	141
Leveraging Text-to-Speech Services	145
Testing Application Accessibility	147
Summary	147
Quiz Questions	148
Exercises	148
References and More Information	148
10 Development Best Practices for Tablets, TVs, and Wearables	151
Understanding Device Diversity	151
Don't Make Assumptions about Device Characteristics	151
Designing Flexible User Interfaces	152
Attracting New Types of Users	153
Leveraging Alternative Resources	153
Using Screen Space Effectively on Big Landscape Screens	153
Developing Applications for Tablets	154
Developing Applications for TV	155
Working with Google TV	156
Google TV Variations	156
Developing Applications for Wearables	158
Summary	159

Quiz Questions	159
Exercises	159
References and More Information	160

III: Leveraging Common Android APIs 161

11 Using Android Networking APIs 163

Understanding Mobile Networking Fundamentals	163
Understanding <code>StrictMode</code> with Networking	164
Accessing the Internet (HTTP)	164
Reading Data from the Web	164
Using <code>URLConnection</code>	165
Parsing XML from the Network	166
Handling Network Operations Asynchronously	167
Retrieving Android Network Status	171
Summary	173
Quiz Questions	173
Exercises	174
References and More Information	174

12 Using Android Web APIs 175

Browsing the Web with <code>WebView</code>	175
Designing a Layout with a <code>WebView</code> Control	176
Loading Content into a <code>WebView</code> Control	176
Adding Features to the <code>WebView</code> Control	178
Managing <code>WebView</code> State	181
Building Web Extensions	182
Browsing the WebKit APIs	182
Extending Web Application Functionality to Android	182
Debugging WebViews with Chrome DevTools	187
Working with Adobe AIR and Flash	187
Summary	188
Quiz Questions	188
Exercises	189
References and More Information	189

13 Using Android Multimedia APIs 191

Working with Multimedia	191
-------------------------	-----

Working with the Camera	192
Capturing Still Images Using the Camera	192
Configuring Camera Mode Settings	196
Working with Common Camera Parameters	197
Zooming the Camera	197
Sharing Images	198
Assigning Images as Wallpapers	199
Choosing among Various Device Cameras	199
Working with Video	200
Recording Video	200
Playing Video	202
Working with Face Detection	203
Working with Audio	204
Recording Audio	204
Playing Audio	205
Sharing Audio	206
Searching for Multimedia	207
Working with Ringtones	208
Introducing the Media Router	209
Summary	209
Quiz Questions	209
Exercises	210
References and More Information	210
14 Using Android Telephony APIs	211
Working with Telephony Utilities	211
Gaining Permission to Access Phone State Information	212
Requesting Call State	212
Requesting Service Information	214
Monitoring Signal Strength and Data Connection Speed	214
Working with Phone Numbers	215
Using SMS	216
Default Messaging Application	216
SMS Provider	217
SMS Applications Other than the Default	217

Making and Receiving Phone Calls	220
Making Phone Calls	220
Receiving Phone Calls	221
Working with SIP	222
Summary	223
Quiz Questions	223
Exercises	223
References and More Information	224
15 Accessing Android's Hardware Sensors	225
Interacting with Device Hardware	225
Using the Device Sensors	226
Working with Different Sensors	226
Configuring the Android Manifest File for Sensors	227
Acquiring a Reference to a Sensor	227
Reading Sensor Data	228
Calibrating Sensors	229
Determining Device Orientation	230
Finding True North	230
Sensor Event Batching	230
Monitoring the Battery	231
Summary	232
Quiz Questions	233
Exercises	234
References and More Information	234
16 Using Android's Optional Hardware APIs	235
Working with Bluetooth	235
Checking for the Existence of Bluetooth Hardware	236
Enabling Bluetooth	237
Querying for Paired Devices	237
Discovering Devices	237
Establishing Connections between Devices	238
Working with USB	239
Working with USB Accessories	240
Working as a USB Host	241

Working with Android Beam	241
Enabling Android Beam Sending	241
Receiving Android Beam Messages	243
Configuring the Manifest File for Android Beam	244
Android Beam over Bluetooth	245
Introducing Host Card Emulation	245
Working with Wi-Fi	245
Introducing Wi-Fi Direct	245
Monitoring Wi-Fi State	246
Summary	248
Quiz Questions	248
Exercises	249
References and More Information	249

IV: Leveraging Google APIs 251

17 Using Location and Map APIs 253

Incorporating Android Location APIs	253
Using the Global Positioning System (GPS)	254
Geocoding Locations	256
Doing More with Android Location-Based Services	260
Incorporating Google Location Services APIs	260
Locating with the Fused Location Provider	260
Doing More with Google Location Services	261
Incorporating Google Maps Android API v2	262
Mapping Locations	263
Summary	268
Quiz Questions	268
Exercises	269
References and More Information	269

18 Working with Google Cloud Messaging 271

An Overview of GCM	271
Understanding GCM Message Flow	272
Understanding the Limitations of the GCM Service	272
Signing Up for GCM	273

Incorporating GCM into Your Applications	273
Exploring the GCM Sample Applications	274
What Alternatives to GCM Exist?	274
Summary	275
Quiz Questions	275
Exercises	275
References and More Information	276
19 An Overview of In-App Billing APIs for Android	277
What Is In-App Billing?	277
Using In-App Billing	278
Leveraging Google Play In-App Billing APIs	279
Leveraging Amazon Appstore for Android In-App Purchasing APIs	280
Leveraging PayPal Billing APIs	280
Leveraging Other Billing APIs	280
Summary	280
Quiz Questions	281
Exercises	281
References and More Information	281
20 Enabling Application Statistics with Google Analytics	283
Creating a Google Account for Analytics	283
Adding the Library to Your Android IDE Project	286
Collecting Data from Your Applications	287
Logging Different Events	287
Using the Google Analytics Dashboard	288
Gathering E-commerce Information	290
Logging E-commerce Events in Your Applications	290
Reviewing E-commerce Reports	291
Tracking Ad and Market Referrals	292
Gathering Statistics	292
Protecting Users' Privacy	293
Summary	293
Quiz Questions	293
Exercises	294
References and More Information	294

21 An Overview of Google Play Game Services 295

- Getting Up and Running with Google Play Game Services 295
- Incorporating Google Play Game Services into Your Applications 296
- Understanding Achievements 297
- Understanding Leaderboards 298
- Saving Game Data with Cloud Save 299
- Introducing Multiplayer Gaming 299
- Understanding Antipiracy 299
- Summary 300
- Quiz Questions 300
- Exercises 300
- References and More Information 301

V: Drawing, Animations, and Graphics Programming with Android 303**22 Developing Android 2D Graphics Applications 305**

- Drawing on the Screen 305
 - Working with Canvases and Paints 305
 - Understanding the Canvas Object 307
 - Understanding the Paint Object 307
- Working with Text 310
 - Using Default Fonts and Typefaces 310
 - Using Custom Typefaces 310
 - Measuring Text Screen Requirements 312
- Working with Bitmaps 312
 - Drawing Bitmap Graphics on a Canvas 313
 - Scaling Bitmap Graphics 313
 - Transforming Bitmaps Using Matrixes 313
 - Bitmap Performance Optimizations 314
- Working with Shapes 315
 - Defining Shape Drawables as XML Resources 315
 - Defining Shape Drawables Programmatically 316
 - Drawing Different Shapes 317
- Leveraging Hardware Acceleration Features 324
 - Controlling Hardware Acceleration 325
 - Fine-Tuning Hardware Acceleration 325

Summary	326
Quiz Questions	326
Exercises	326
References and More Information	327
23 Working with Animation	329
Animating Your Applications	329
Working with Drawable Animation	329
Working with View Animations	331
Working with Property Animation	336
Working with Different Interpolators	341
Animating <code>Activity</code> Launch	341
State Animations with Scenes and Transitions	342
Summary	342
Quiz Questions	342
Exercises	343
References and More Information	343
24 Developing Android 3D Graphics Applications	345
Working with OpenGL ES	345
Leveraging OpenGL ES in Android	346
Ensuring Device Compatibility	346
Using OpenGL ES APIs in the Android SDK	347
Handling OpenGL ES Tasks Manually	347
Creating a <code>SurfaceView</code>	348
Starting Your OpenGL ES Thread	349
Initializing EGL	350
Initializing GL	352
Drawing on the Screen	353
Drawing 3D Objects	353
Drawing Your Vertices	353
Coloring Your Vertices	355
Drawing More Complex Objects	356
Lighting Your Scene	358
Texturing Your Objects	359
Interacting with Android Views and Events	362
Enabling the OpenGL Thread to Talk to the Application Thread	362

Enabling the Application Thread to Talk to the OpenGL Thread	363
Cleaning Up OpenGL ES	365
Using <code>GLSurfaceView</code> (Easy OpenGL ES)	366
Using OpenGL ES 2.0	369
Configuring Your Application for OpenGL ES 2.0	369
Requesting an OpenGL ES 2.0 Surface	370
Exploring OpenGL ES 3.0	373
Summary	374
Quiz Questions	374
Exercises	374
References and More Information	375
25 Using the Android NDK	377
Determining When to Use the Android NDK	377
Installing the Android NDK	378
Exploring the Android NDK Sample Application	379
Creating Your Own NDK Project	379
Calling Native Code from Java	380
Handling Parameters and Return Values	381
Using Exceptions with Native Code	382
Using Native Activities	384
Improving Graphics Performance	384
Comparing <code>RenderScript</code> to the NDK	385
Computing with <code>RenderScript</code>	385
Native <code>RenderScript</code>	385
Summary	386
Quiz Questions	386
Exercises	386
References and More Information	386
VI: Maximizing Android's Unique Features	389
26 Extending Android Application Reach	391
Enhancing Your Applications	391
Working with App Widgets	392
Creating an App Widget	393
Installing an App Widget to the Home Screen	400

Becoming an App Widget Host	401
Introducing Lock Screen App Widgets	401
Installing an App Widget to the Lock Screen	403
Working with Live Wallpapers	404
Creating a Live Wallpaper	404
Creating a Live Wallpaper <i>Service</i>	404
Creating a Live Wallpaper Configuration	406
Configuring the Android Manifest File for Live Wallpapers	406
Installing a Live Wallpaper	407
Introducing Daydream	408
Acting as a Content Type Handler	410
Determining <i>Intent</i> Actions and MIME Types	411
Implementing the <i>Activity</i> to Process the Intents	412
Registering the Intent Filter	412
Summary	413
Quiz Questions	413
Exercises	414
References and More Information	414
27 Enabling Application Search	415
Making Application Content Searchable	415
Enabling Searches in Your Application	416
Creating a Search Configuration	417
Creating a Search <i>Activity</i>	422
Configuring the Android Manifest File for Search	423
Enabling Global Search	424
Updating a Search Configuration for Global Searches	425
Updating Search Settings for Global Searches	425
Summary	426
Quiz Questions	426
Exercises	426
References and More Information	426

28	Managing User Accounts and Synchronizing User Data	427
	Managing Accounts with the Account Manager	427
	Multiple Users, Restricted Profiles, and Accounts	428
	Synchronizing Data with Sync Adapters	429
	Using Backup Services	430
	Choosing a Remote Backup Service	430
	Implementing a Backup Agent	431
	Backing Up and Restoring Application Data	434
	Summary	435
	Quiz Questions	436
	Exercises	436
	References and More Information	436

VII: Advanced Topics in Application Publication and Distribution 437

29	Internationalizing Your Applications	439
	Localizing Your Application's Language	439
	Internationalization Using Alternative Resources	439
	Changing the Language Settings	442
	Implementing Locale Support Programmatically	444
	Right-to-Left Language Localization	445
	Translation Services through Google Play	445
	Using the Developer Console	446
	Publishing Applications for Foreign Users	446
	Summary	446
	Quiz Questions	446
	Exercises	447
	References and More Information	447
30	Protecting Applications from Software Piracy	449
	All Applications Are Vulnerable	449
	Using Secure Coding Practices	450

Obfuscating with ProGuard	450
Configuring ProGuard for Your Android Applications	451
Dealing with Error Reports after Obfuscation	452
Leveraging the License Verification Library	452
Other Antipiracy Tips	453
Summary	454
Quiz Questions	454
Exercises	455
References and More Information	455

VIII: Preparing for Future Android Releases 457

31 Introducing the L Developer Preview 459

Exploring the L Developer Preview	459
Improving Performance	460
Improving the User Experience	461
Introducing Android TV	464
Understanding Android TV Development Requirements	464
Understanding TV Application Hardware Limitations	465
Summary	465
Quiz Questions	465
Exercises	466
References and More Information	466

IX: Appendixes 467

A Quick-Start Guide: Android Debug Bridge 469

Listing Connected Devices and Emulators	469
Directing ADB Commands to Specific Devices	470
Starting and Stopping the ADB Server	470
Stopping the ADB Server Process	470
Starting and Checking the ADB Server Process	470
Listing ADB Commands	470
Issuing Shell Commands	471
Issuing a Single Shell Command	471
Using a Shell Session	471
Using the Shell to Start and Stop the Emulator	471

Copying Files	472
Sending Files to a Device or Emulator	472
Retrieving Files from a Device or Emulator	472
Installing and Uninstalling Applications	473
Installing Applications	473
Reinstalling Applications	473
Uninstalling Applications	473
Working with LogCat Logging	474
Displaying All Log Information	474
Including Date and Time with Log Data	474
Filtering Log Information	474
Clearing the Log	476
Redirecting Log Output to a File	476
Accessing the Secondary Logs	476
Controlling the Backup Service	476
Forcing Backup Operations	477
Forcing Restore Operations	477
Wiping Archived Data	477
Generating Bug Reports	477
Using the Shell to Inspect SQLite Databases	478
Using the Shell to Stress Test Applications	478
Letting the Monkey Loose on Your Application	478
Listening to Your Monkey	478
Directing Your Monkey's Actions	479
Training Your Monkey to Repeat His Tricks	480
Keeping the Monkey on a Leash	480
Learning More about Your Monkey	481
Installing Custom Binaries via the Shell	481
Summary	482
Quiz Questions	482
Exercises	483
References and More Information	483
B Quick-Start Guide: SQLite	485
Exploring Common Tasks with SQLite	485
Using the <code>sqlite3</code> Command-Line Interface	486
Launching the ADB Shell	486
Connecting to a SQLite Database	486

Exploring Your Database	487
Importing and Exporting the Database and Its Data	488
Executing SQL Commands on the Command Line	490
Using Other <code>sqlite3</code> Commands	490
Understanding SQLite Limitations	490
Learning by Example: A Student Grade Database	491
Designing the Student Grade Database Schema	491
Creating Simple Tables with <code>AUTOINCREMENT</code>	492
Inserting Data into Tables	492
Querying Tables for Results with <code>SELECT</code>	493
Using Foreign Keys and Composite Primary Keys	493
Altering and Updating Data in Tables	495
Querying Multiple Tables Using <code>JOIN</code>	495
Using Calculated Columns	496
Using Subqueries for Calculated Columns	497
Deleting Tables	497
Summary	497
Quiz Questions	498
Exercises	498
References and More Information	498
C Java for Android Developers	499
Learning the Java Programming Language	499
Learning the Java Development Tools	499
Familiarizing Yourself with Java Documentation	500
Understanding Java Shorthand	500
Chaining Methods and Unnecessary Temp Variables	501
Looping Infinitely	501
Working with Unary and Ternary Operators	502
Working with Inner Classes	503
Summary	505
Quiz Questions	505
Exercises	505
References and More Information	505

D Quick-Start Guide: Android Studio	507
Getting Up and Running with Android Studio	507
Launching Android Studio for the First Time	508
Configuring Android Studio	508
Creating an Android Studio Project	509
Understanding the Android Studio Project Structure	512
Learning about the Gradle Build System	513
Overview of the Android Studio User Interface	513
Introducing the Layout Editor	513
Working in Design View	514
Working in Text View	514
Using the Preview Controls	515
Debugging Your Android Studio Applications	515
Setting Breakpoints	515
Stepping through Code	516
Useful Keyboard Shortcuts	517
Summary	517
Quiz Questions	517
Exercises	518
References and More Information	518
E Answers to Quiz Questions	519
Index	527

This page intentionally left blank

Acknowledgments

This book is the result of collaboration among a great group, from the efforts of the team at Pearson Education (Addison-Wesley), from the suggestions made by the technical reviewers, and from the support of family, friends, coworkers, and acquaintances alike. We'd like to thank the Android developer community, Google, and the Open Handset Alliance for their vision and expertise. Special thanks go to Mark Taub for believing in the vision for this edition; Laura Lewin, who was the driving force behind the book—without her this book would not have become a reality; Olivia Basegio, who was instrumental in orchestrating the efforts of everyone involved; Songlin Qiu for performing countless iterations combing through the manuscript and making this book ready for production; and the technical reviewers: Doug Jones who suggested improvements of the fine details, Ray Rischpater, who made many beneficial recommendations, and Valerie Shipbaugh who spotted areas in need of clarification (as well as Mike Wallace, Mark Gjoel, Dan Galpin, Tony Hillerson, Ronan Schwarz, and Charles Stearns, who reviewed previous editions and incarnations of this book). Dan Galpin also graciously provided the clever Android graphics used for Tips, Notes, and Warnings. We also thank Hans Bodlaender for letting us use the nifty chess font he developed as a hobby project.

This page intentionally left blank

About the Authors

Joseph Annuzzi, Jr., is a freelance software architect, graphic artist, inventor, entrepreneur, and author. He usually can be found mastering the Android platform, implementing cutting-edge HTML5 capabilities, leveraging various cloud technologies, speaking in different programming languages, working with diverse frameworks, integrating with various social APIs, tinkering with peer-to-peer, cryptography, and biometric algorithms, or creating stunningly realistic 3D renders. He is always on the lookout for disruptive Internet and mobile technologies and has multiple patent applications in process. He graduated from the University of California, Davis, with a BS in managerial economics and a minor in computer science and lives where much of the action is: Silicon Valley.

When he is not working with technology, he has been known to lounge in the sun on the beaches of the Black Sea with international movie stars; he has trekked through the Bavarian forest in winter, has immersed himself in the culture of the Italian Mediterranean, and has narrowly escaped the wrath of an organized crime ring in Eastern Europe after his taxi dropped him off in front of the bank ATM they were liquidating. He also lives an active and healthy lifestyle, designs and performs custom fitness training routines to stay in shape, and adores his loyal beagle, Cleopatra.

Lauren Darcey is responsible for the technical leadership and direction of a small software company specializing in mobile technologies, including Android and iOS consulting services. With more than two decades of experience in professional software production, Lauren is a recognized authority in application architecture and the development of commercial-grade mobile applications. Lauren received a BS in computer science from the University of California, Santa Cruz.

She spends her copious free time traveling the world with her geeky mobile-minded husband and pint-sized geekling daughter. She is an avid nature photographer. Her work has been published in books and newspapers around the world. In South Africa, she dove with 4-meter-long great white sharks and got stuck between a herd of rampaging hippopotami and an irritated bull elephant. She's been attacked by monkeys in Japan, gotten stuck in a ravine with two hungry lions in Kenya, gotten thirsty in Egypt, narrowly avoided a coup d'état in Thailand, geocached her way through the Swiss Alps, drunk her way through the beer halls of Germany, slept in the crumbling castles of Europe, and gotten her tongue stuck to an iceberg in Iceland (while being watched by a herd of suspicious wild reindeer). Most recently, she can be found hiking along the Appalachian Trail with her daughter and documenting the journey with Google Glass.

Shane Conder has extensive application development experience and has focused his attention on mobile and embedded development for well over a decade. He has designed and developed many commercial applications for Android, iOS, BREW, BlackBerry, J2ME, Palm, and Windows Mobile—some of which have been installed on millions of phones worldwide. Shane has written extensively about the tech industry and is known for his keen insights regarding mobile development platform trends. Shane received a BS in computer science from the University of California, Santa Cruz.

A self-admitted gadget freak, Shane always has the latest smartphone, tablet, or wearable. He enjoys traveling the world with his geeky wife, even if she did make him dive with 4-meter-long great white sharks and almost get eaten by a lion in Kenya. He admits that he has to take at least three devices with him when backpacking (“just in case”)—even where there is no coverage. Lately, his smart watch collection has exceeded his number of wrists. Luckily, his young daughter is happy to offer her own. Such are the burdens of a daughter of engineers.

Introduction

Android is a popular, free, and open-source mobile platform that has taken the wireless world by storm. This book and *Introduction to Android™ Application Development: Android Essentials, Fourth Edition*, provide comprehensive guidance for software development teams on designing, developing, testing, debugging, and distributing professional Android applications. If you're a veteran mobile developer, you can find tips and tricks to streamline the development process and take advantage of Android's unique features. If you're new to mobile development, these books provide everything you need to make a smooth transition from traditional software development to mobile development—specifically, its most promising platform: Android.

Who Should Read This Book?

This book includes tips for successful mobile development based upon our years in the mobile industry, and it covers everything you need to know to run a successful Android project from concept to completion. We cover how the mobile software process differs from traditional software development, including tricks to save valuable time and pitfalls to avoid. Regardless of the size of your project, this book is for you.

This book was written for various audiences:

- **Software developers who want to learn to develop professional Android applications.** The bulk of this book is targeted at software developers with Java experience who do not necessarily have mobile development experience. More seasoned developers of mobile applications can learn how to take advantage of Android and how it differs from the other technologies on the mobile development market today.
- **Other audiences.** This book is useful not only to software developers, but also to corporations looking at potential vertical market applications, entrepreneurs thinking about cool phone applications, and hobbyists looking for some fun with their new phones. Businesses seeking to evaluate Android for their specific needs (including feasibility analysis) can also find the information provided valuable. Anyone with an Android handset and a good idea for a mobile application can put the information in this book to use for fun and profit.

How This Book Is Structured

Advanced Android™ Application Development, Fourth Edition, focuses on advanced Android topics, including leveraging various Android application programming interfaces (APIs)

for threading, networking, location-based services, hardware sensors, animation, graphics, and more. Coverage of advanced Android application components, such as services, application databases, content providers, and intents, is also included. Developers will learn to design advanced user interface (UI) components and integrate their applications deeply into the platform. Finally, developers will learn how to extend their applications beyond traditional boundaries using optional features of the Android platform, including the Android Native Development Kit (NDK), Google Cloud Messaging (GCM), Google Play In-app Billing APIs, Google Analytics APIs, Android Wear, Google Play game services, and more.

Advanced Android™ Application Development, Fourth Edition, is divided into nine parts. Here is an overview of the various parts:

■ **Part I: Advanced Android Application Design Principles**

Part I picks up where *Introduction to Android™ Application Development: Android Essentials, Fourth Edition*, leaves off in terms of application design techniques. We begin by talking about asynchronous processing. We then move on to some of the more complex Android application components, such as services, application databases (SQLite), content providers, intents, and notifications.

■ **Part II: Advanced Android User Interface Design Principles**

Part II dives deeper into some of the more advanced user interface tools and techniques available as part of the Android Software Development Kit (SDK), including working with action bars, gathering input through nonstandard methods such as gestures and voice recognition, and much more. You will also learn more about how to develop applications that are accessible to different types of users with impairments.

■ **Part III: Leveraging Common Android APIs**

Part III dives deeper into some of the more advanced and specialty APIs available as part of the Android SDK, including networking, web APIs, multimedia (including the camera), telephony, and hardware sensors.

■ **Part IV: Leveraging Google APIs**

Part IV is for those developers who need to integrate with the many available features provided by Google. We cover Google location services, Google Maps Android services, Google Cloud Messaging, Google In-app Billing, Google Analytics, and Google Play game services.

■ **Part V: Drawing, Animations, and Graphics Programming with Android**

Part V is for those developers incorporating graphics of any kind into their applications. We cover both 2D and 3D graphics (OpenGL ES), animation, and the Android NDK.

■ **Part VI: Maximizing Android's Unique Features**

Part VI discusses some of the many ways the Android platform is different from other mobile platforms and how your applications can leverage its unique features. Here you will learn how to extend your application features beyond the traditional

borders of mobile applications, integrating them with the Android operating system. App Widgets, enabling searches, and backups are just some of the topics discussed.

■ **Part VII: Advanced Topics in Application Publication and Distribution**

Part VII covers more advanced topics in application publication and distribution, including how to internationalize your applications and taking measures to protect your intellectual property from software pirates.

■ **Part VIII: Preparing for Future Android Releases**

Part VIII introduces the newest version of the Android SDK, the L Developer Preview. We highlight many of the most anticipated features available in this release, including Android Runtime (ART), Project Volta, material design, and Android TV.

■ **Part IX: Appendixes**

Part IX includes a helpful quick-start guide for the Android Debug Bridge (ADB) tool, a refresher course on using SQLite, and a quick-start guide for the Android Studio IDE. There is also an appendix discussing Java for Android developers and one dedicated to providing answers to the quiz questions.

Key Questions Answered in This Book

This book answers the following questions:

1. How can developers write responsive applications?
2. How are Android applications structured? How are background operations handled with services? What are broadcast intents and how can applications use them effectively?
3. How do applications store data persistently using SQLite? How can applications act as content providers and why would they want to do so?
4. How do applications interact with the Android operating system? How do applications trigger system notifications, access underlying device hardware, and monitor device sensors?
5. How can developers design the best user interfaces for the devices of today and tomorrow? How can developers work with 2D and 3D graphics and leverage animation opportunities on Android?
6. How can developers write high-performance, computationally intensive applications using native code?
7. What are some of the most commonly used APIs for networking, location services, maps, multimedia, telephony, and Internet access?
8. What do managers, developers, and testers need to look for when planning, developing, and testing a mobile development application?
9. How do mobile teams design bulletproof Android applications for publication?

10. How can developers make their applications leverage everything Android has to offer in the form of App Widgets, live wallpapers, and other system perks?
11. How can applications take advantage of some of the optional third-party APIs available for use, such as Google Play's In-app Billing and License Verification Library, Google Analytics, Google Play game services, Google location services, Google Maps Android v2 services, and Google Cloud Messaging services?
12. How can developers make use of new Android preview features such as the new Android Studio or Android Wear?

An Overview of Changes in This Edition

When we began writing the first edition of this book, there were no Android devices on the market. Today there are hundreds of types of devices shipping all over the world—smartphones, tablets, e-book readers, smart watches, and specialty devices such as gaming consoles, Google TV, and Google Glass.

The Android platform has gone through extensive changes since the first edition of this book was published. The Android SDK has many new features, and the development tools have received much-needed upgrades. Android, as a technology, is now on solid footing in the mobile marketplace.

For this new edition, we took the opportunity to add content covering the latest and greatest features Android has to offer—but don't worry, it's still the book readers loved the first, second, and third time around; it's just bigger, better, and more comprehensive. In addition to adding new content, we've retested and upgraded all existing content (text and sample code) for use with the latest Android SDKs available while still remaining backward compatible. We created quiz questions to help readers ensure that they understand each chapter's content, and we added end-of-chapter exercises for readers to perform to dig deeper into all that Android has to offer. The Android development community is diverse, and we aim to support all developers, regardless of which devices they are developing for. This includes developers who need to target nearly all platforms, so coverage in some key areas of older SDKs continues to be included as it's often the most reasonable option for compatibility.

Here are some of the highlights of the additions and enhancements we've made in this edition:

- Coverage of the latest and greatest Android tools and utilities is included.
- The chapter on content providers has been rewritten with updated code samples in reference to a simpler application.
- The chapter on notifications has been rewritten to include a new application and code samples demonstrating how to create notifications with the new `NotificationCompat.Builder()` class, and we show how to create expandable and contractible notifications.
- There are totally new chapters that cover Google Cloud Messaging and Google Play game services, and a new appendix that shows you how to get up and running quickly with Android Studio.

- The chapter about location and map APIs has been rewritten to include the new Google location services APIs and the Google Maps Android v2 APIs, allowing you to build even more compelling location services into your applications.
- The chapter on Google Analytics has been rewritten and includes a new application with updated code demonstrating how to make use of the latest version of the Google Analytics SDK for Android.
- The telephony chapter includes information describing the latest changes that affect Short Message Service (SMS) applications, discussing the behavioral differences between the default SMS app and the nondefault SMS apps.
- We've added coverage of hot topics such as Android Wear, sensor event batching, state animations with scenes and transitions, OpenGL ES 3.0, Lock screen App Widgets, Daydream, and Google Play App Translation Service.
- All chapters and appendixes now include quiz questions and exercises for readers to test their knowledge of the subject matter presented.
- All existing chapters have been updated, often with entirely new sections.
- All sample code and accompanying applications have been updated to work with the latest SDK.

As you can see, we cover many of the hottest and most exciting features that Android has to offer. We didn't take this revision lightly; we touched every existing chapter, updated content, and added new chapters as well. Finally, we included many additions, clarifications, and, yes, even a few fixes based upon the feedback from our fantastic (and meticulous) readers. Thank you!

The Development Environment Used in This Book

The Android code in this book was written using the following development environments:

- Windows 7, Windows 8, and Mac OS X 10.9.x
- Android ADT Bundle (20140321 files were used)
- Android Studio (135.1078000 files were used)
- Android SDK Version 4.4, API Level 19 (KitKat)
- Android SDK Tools Revision 22.6.4
- Android SDK Platform Tools 19.0.2
- Android SDK Build Tools 19.1
- Android Support Library Revision 19.1 (where applicable)
- Google Analytics App Tracking SDK version 4 (where applicable)
- Google Play services Revision 17 (where applicable)
- Android NDK r9d (the `android-ndk-r9d-windows-x86.zip` file was used)

- Java SE Development Kit (JDK) 6 Update 45, and JDK 7 Update 55
- Android devices: Nexus 4 and 5 (phones), Nexus 7 (first- and second-generation 7-inch tablet), Nexus 10 (10-inch tablet), including various other devices and form factors

The Android platform continues to grow in market share against competing mobile platforms, such as Apple iOS and BlackBerry. New and exciting types of devices reach consumers' hands at a furious pace. Developers have embraced Android as a target platform to reach the device users of today and tomorrow.

Android's latest major platform update, Android 4.4, frequently called by its code name KitKat, has many new features that help differentiate Android from the competition. This book features the latest SDK and tools available, but it does not focus on them to the detriment of popular legacy versions of the platform. The book is meant to be an overall reference to help developers support all popular devices on the market today. As of the writing of this book, a sizable proportion of users (23.9 percent) have devices that run Android 4.3 or 4.4. Of course, some devices receive upgrades, and users purchase new devices as they become available, but for now, developers need to straddle this gap and support numerous versions of Android to reach the majority of users in the field. In addition, the next version of the Android operating system is likely to be released in the near future.

So what does this mean for this book? It means we provide legacy API support and discuss some of the newer APIs available only in later versions of the Android SDK. We discuss strategies for supporting all (or at least most) users in terms of compatibility. And we provide screenshots that highlight different versions of the Android SDK, because each major revision has brought with it a change in the look and feel of the overall platform. That said, we are assuming that you are downloading the latest Android tools, so we provide screenshots and steps that support the latest tools available at the time of writing, not legacy tools. Those are the boundaries we set when trying to determine what to include or leave out of this book.

Supplementary Materials Available

The source code is also available for download from our book's website: <http://advancedandroidbook.blogspot.com/2014/07/book-code-samples.html>.

Where to Find More Information

There is a vibrant, helpful Android developer community on the Web. Here are a number of useful websites for Android developers and followers of the wireless industry:

- **Android Developer website:** The Android SDK and developer reference site:
<http://developer.android.com/>
- **Google Plus:** Android Developers Group:
<https://plus.google.com/+AndroidDevelopers/posts>

- **YouTube:** Android Developers channels:
<http://youtube.com/user/androiddevelopers>
- **Stack Overflow:** The Android website with great technical information (complete with tags) and an official support forum for developers:
<http://stackoverflow.com/questions/tagged/android>
- **Open Handset Alliance:** Android manufacturers, operators, and developers:
<http://openhandsalliance.com/>
- **Google Play:** Buy and sell Android applications:
<https://play.google.com/store>
- **Mobiletuts+:** Mobile development tutorials, including Android:
<http://code.tutsplus.com/categories/android-sdk>
- **Android Tools Project Site:** The tools team discusses updates and changes:
<https://sites.google.com/a/android.com/tools/recent>
- **FierceDeveloper:** A weekly newsletter for wireless developers:
<http://fiercedev.com/>
- **XDA-Developers Android forum:** From general development to ROMs:
<http://forum.xda-developers.com/android>
- **Developer.com:** A developer-oriented site with mobile articles:
<http://developer.com/>

Conventions Used in This Book

This book uses the following conventions:

- Code, directory paths, and programming terms are set in monospace font.
- Java import statements, exception handling, and error checking are often removed from printed code samples for clarity and to keep the book at a reasonable length.

This book also presents information in the following types of sidebars:



Tip

— Tips provide useful information or hints related to the current text.



Note

— Notes provide additional information that might be interesting or relevant.



Warning

- Warnings provide hints or tips about pitfalls that may be encountered and how to avoid them.

Contacting the Authors

We welcome your comments, questions, and feedback. We invite you to visit our blog at:

- <http://advancedandroidbook.blogspot.com>

Or, email us at:

- advancedandroidbook4e@gmail.com

Circle us on Google+:

- Joseph Annuzzi, Jr.: <http://goo.gl/FBQeL>
- Lauren Darcey: <http://goo.gl/P3RGo>
- Shane Conder: <http://goo.gl/BpVJh>

Handling Advanced User Input

Users interact with Android devices in many ways, including using keyboards, touch-screen gestures, and even voice. Different devices support different input methods and have different hardware. For example, certain devices have hardware keyboards, and others rely only on software keyboards. In this chapter, you will learn about the different input methods available to developers and how you can use them to great effect within your applications.

Working with Textual Input Methods

The Android SDK includes input method framework classes that enable interested developers to use powerful input methods and create their own input methods, such as custom software keyboards and other Input Method Editors (IMEs). Users can download custom IMEs to use on their devices. For example, there's nothing stopping a developer from creating a custom keyboard with *Lord of the Rings*-style Elvish characters, smiley faces, or Greek symbols.



Tip

Most device settings related to input methods are available under the Settings, Language & input menu. Here, users can select the language, configure the custom user dictionary, and make changes to how their keyboards function.

The Android SDK also includes a number of other text input utilities that might benefit application users, such as text prediction, dictionaries, and the clipboard framework, which can be used to enable sophisticated cut-and-paste features in your application for text and much more.

Working with Software Keyboards

Because text input methods are locale-based (different countries use different alphabets and keyboards) and situational (numeric versus alphabetic versus special keys), the Android platform has trended toward software keyboards as opposed to relying on hardware manufacturers to deliver specialized hardware keyboards.

Choosing the Appropriate Software Keyboard

The Android platform has a number of software keyboards available for use. One of the easiest ways to enable your users to enter data efficiently is to specify the type of input expected in each text input field.



Tip

- Many of the code examples provided in this section are taken from the `SimpleTextInputTypes` application. The source code for this application is provided for download on the book's website.

For example, to specify an `EditText` that should take only capitalized textual input, you can set the `inputType` attribute as follows:

```
<EditText
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:inputType="text|textCapCharacters">
</EditText>
```

Figure 8.1 shows a number of `EditText` controls with different `inputType` configurations.

The input type dictates which software keyboard is used by default, and it enforces appropriate rules, such as limiting input to certain characters. Figure 8.2 (left) illustrates what the software keyboard looks like for an `EditText` control with its `inputType` attribute set to all capitalized text input. Note that the software keyboard keys are all capitalized. If you were to set the `inputType` to `textCapWords` instead, the keyboard would switch to lowercase after the first letter of each word and then back to uppercase after a space. Figure 8.2 (middle) illustrates what the software keyboard looks like for an `EditText` control with its `inputType` attribute set to number. Figure 8.2 (right) illustrates what the software keyboard looks like for an `EditText` control with its `inputType` attribute set to textual input, where each sentence begins with a capital letter and the text can be multiple lines.

Depending on the user's keyboard settings (specifically, if the user has enabled the Show correction suggestions and Auto-correction options in the Android Keyboard settings of the device), the user might also see suggested words or spelling fixes while typing. For a complete list of `inputType` attribute values and their uses, see <http://developer.android.com/reference/android/R.attr.html#inputType>.



Tip

- You can also have your `Activity` react to the display of software keyboards (to adjust where fields are displayed, for example) by requesting the `WindowManager` as a system `Service` and modifying the layout parameters associated with the `softInputMode` field.

For more fine-tuned control over input methods, see the `android.view.inputmethod.InputMethodManager` class.

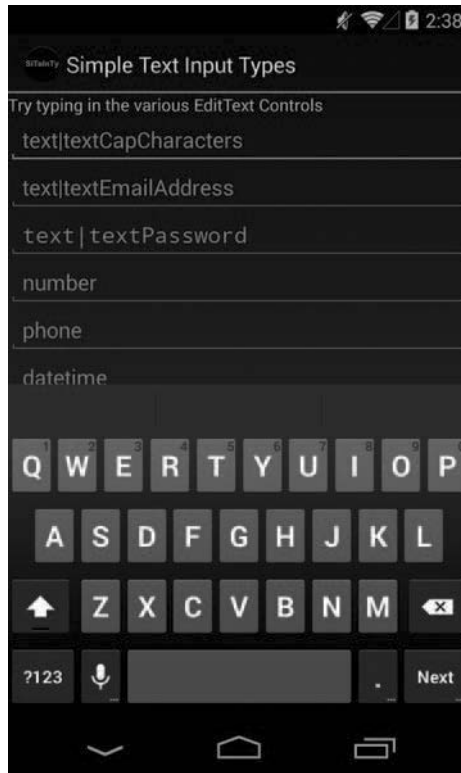


Figure 8.1 EditText controls with different input types.

Providing Custom Software Keyboards

If you are interested in developing your own software keyboards, we highly recommend the following references:

- IMEs are implemented as an Android `Service`. Begin by reviewing the Android packages called `android.inputmethodservice` and `android.view.inputmethod`, which can be used to implement custom input methods.
- The `SoftKeyboard` legacy sample application in the Android SDK provides an implementation of a software keyboard.
- The Android Developers Blog has articles on on-screen input methods (<http://android-developers.blogspot.com/2009/04/creating-input-method.html>) and creating an input method (<http://android-developers.blogspot.com/2009/04/creating-input-method.html>). Don't forget to add voice typing to your input method (<http://android-developers.blogspot.com/2011/12/add-voice-typing-to-your-ime.html>).

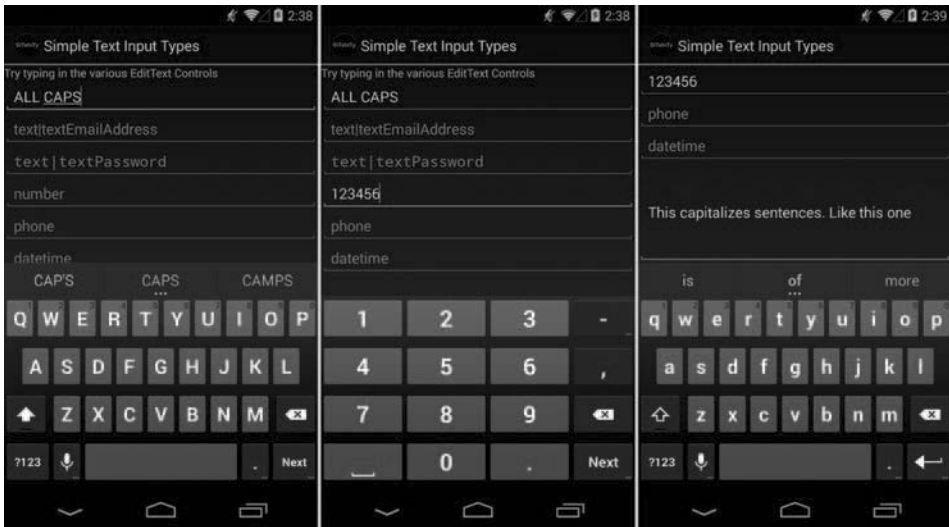


Figure 8.2 The software keyboards associated with specific input types.

Working with Text Prediction and User Dictionaries

Text prediction is a powerful and flexible feature that is available on Android devices. We’ve already talked about many of these technologies in other parts of this book, but they merit mentioning in this context as well:

- In *Introduction to Android Application Development: Android Essentials, Fourth Edition*, you learned how to use `AutoCompleteTextView` and `MultiAutoCompleteTextView` controls to help users input common words and strings.
- In Chapter 3, “Leveraging SQLite Application Databases,” you learned how to tie an `AutoCompleteTextView` control to an underlying SQLite database table.
- In *Introduction to Android Application Development: Android Essentials, Fourth Edition*, you learned about the `UserDictionary` content provider (`android.provider.UserDictionary`), which can be used to add words to the user’s custom dictionary of commonly used words.

Using the Clipboard Framework

On Android devices running Android 3.0 and higher (API Level 11), developers can access the clipboard to perform copy and paste actions. Previous to this, the clipboard had no public API. To leverage the clipboard in your applications, you need to use the clipboard

framework of the Android SDK. You can copy and paste different data structures—everything from text to references to files to application shortcuts—as `Intent` objects. The clipboard holds only a single set of clipped data at a time, and the clipboard is shared across all applications, so you can easily copy and paste content between applications.

Copying Data to the System Clipboard

To save data to the system clipboard, call `getSystemService()` and request the clipboard `Service`'s `ClipboardManager` (`android.content.ClipboardManager`). Then, create a `ClipData` (`android.content.ClipData`) object and populate it with the data you want to save to the clipboard. Finally, commit the clip using the `ClipboardManager` class method `setPrimaryClip()`.

Pasting Data from the System Clipboard

To retrieve data from the system clipboard, call `getSystemService()` and request the clipboard `Service`'s `ClipboardManager` (`android.content.ClipboardManager`). You can determine whether the clipboard contains data by using the `hasPrimaryClip()` method. After you have determined whether there is valid data in the system clipboard, you can inspect its description and type and ultimately retrieve the `ClipData` object using the `getPrimaryClip()` method.

Handling User Events

You've seen how to do basic event handling in some of the previous control examples. For instance, you know how to handle when a user clicks on a button. There are a number of other events generated by various actions the user might take. This section briefly introduces you to some of these events. First, though, we need to talk about the input states in Android.

Listening for Touch Mode Changes

The Android screen can be in one of two states. The state determines how the focus on `View` controls is handled. When touch mode is on, typically only objects such as `EditText` get focus when selected. Other objects, because they can be selected directly by the user tapping on the screen, won't take focus but instead trigger their action, if any. When not in touch mode, however, the user can change focus among even more object types. These include buttons and other views that normally need only a click to trigger their action.

Knowing what mode the screen is in is useful if you want to handle certain events. If, for instance, your application relies on the focus or lack of focus on a particular control, your application might need to know whether the device is in touch mode because the focus behavior is likely different.

Your application can register to find out when the touch mode changes by using the `addOnTouchModeChangeListener()` method in the `android.view.ViewTreeObserver` class. Your application needs to implement the `ViewTreeObserver.OnTouchModeChangeListener` class to listen for these events. Here is a sample implementation:

```
View all = findViewById(R.id.events_screen);
ViewTreeObserver vto = all.getViewTreeObserver();
vto.addOnTouchModeChangeListener(
    new ViewTreeObserver.OnTouchModeChangeListener() {
        public void onTouchModeChanged(
            boolean isInTouchMode) {
            events.setText("Touch mode: " + isInTouchMode);
        }
    });
```

In this example, the top-level `View` in the layout is retrieved. A `ViewTreeObserver` listens to a `View` and all its child `View` objects. Using the top-level `View` of the layout means the `ViewTreeObserver` listens to events in the entire layout. An implementation of the `onTouchModeChanged()` method provides the `ViewTreeObserver` with a method to call when the touch mode changes. It merely passes in which mode the `View` is now in.

In this example, the mode is written to a `TextView` named `events`. We use this same `TextView` in further event handling examples to show on the screen which events our application has been told about. The `ViewTreeObserver` can enable applications to listen to a few other events on an entire screen.

By running this sample code, we can demonstrate the touch mode changing to `true` immediately when the user taps on the touchscreen. Conversely, when the user chooses to use any other input method, the application reports that touch mode is `false` immediately after the input event, such as a key being pressed.

Listening for Events on the Entire Screen

You saw in the last section how your application can watch for changes to the touch mode state of the screen using the `ViewTreeObserver` class. The `ViewTreeObserver` also provides other events that can be watched for on a full screen or an entire `View` and all of its children. Some of these are:

- `Draw` or `PreDraw`: Get notified before the `View` and its children are drawn.
- `GlobalLayout`: Get notified when the layout of the `View` and its children might change, including visibility changes.
- `GlobalFocusChange`: Get notified when the focus in the `View` and its children changes.

Your application might want to perform some actions before the screen is drawn. You can do this by calling the method `addOnPreDrawListener()` with an implementation of the `ViewTreeObserver.OnPreDrawListener` class interface or by calling the method

`addOnDrawListener()` with an implementation of the `ViewTreeObserver.OnDrawListener` class interface.

Similarly, your application can find out when the layout or visibility of a `View` has changed. This might be useful if your application dynamically changes the display contents of a `View` and you want to check to see whether a `View` still fits on the screen. Your application needs to provide an implementation of the `ViewTreeObserver.OnGlobalLayoutListener` class interface to the `addOnGlobalLayoutListener()` method of the `ViewTreeObserver` object.

Finally, your application can register to find out when the focus changes between a `View` control and any of its child `View` controls. Your application might want to do this to monitor how a user moves about on the screen. When in touch mode, though, there might be fewer focus changes than when touch mode is not set. In this case, your application needs to provide an implementation of the `ViewTreeObserver.OnGlobalFocusChangeListener` class interface to the `addOnGlobalFocusChangeListener()` method. Here is a sample implementation of this:

```
vto.addOnGlobalFocusChangeListener(new
    ViewTreeObserver.OnGlobalFocusChangeListener() {
        public void onGlobalFocusChanged(
            View oldFocus, View newFocus) {
            if (oldFocus != null && newFocus != null) {
                events.setText("Focus \nfrom: " +
                    oldFocus.toString() + " \nto: " +
                    newFocus.toString());
            }
        }
    });
```

This example uses the same `ViewTreeObserver`, `vto`, and `TextView` events as the previous example. It shows that both the currently focused `View` object and the previously focused `View` object are passed to the listener as method parameters. From here, your application can perform needed actions.

If your application merely wants to check values after the user has modified a particular `View` object, though, you might need to register to listen for focus changes only of that particular `View` object. This is discussed later in this chapter.

Listening for Long Clicks

You can add a context menu or a contextual action bar to a `View` that is activated when the user performs a long click on that `View`. A long click is typically when a user presses on the touchscreen and holds a finger there until an action is performed. However, a long press event can also be triggered if the user navigates there with a non-touch method, such as via a keyboard or a button. This action is also often called a press-and-hold action.

Although the context menu is a great typical use case for the long-click event, you can listen for the long-click event and perform any action you want. However, this is the same event that triggers the context menu. If you've already added a context menu to a `View`, you might not want to listen for the long-click event as other actions or side effects might

confuse the user or even prevent the context menu or contextual action bar from showing. As always with good user interface design, try to be consistent for usability's sake.



Tip

- Usually a long click is an alternative action to a standard click. If a left-click on a computer is the standard click, a long click can be compared to a right-click.

Your application can listen to the long-click event on any `View`. The following example demonstrates how to listen for a long-click event on a `Button` control:

```
Button long_press = (Button) findViewById(R.id.long_press);
long_press.setOnLongClickListener(new View.OnLongClickListener() {
    public boolean onLongClick(View v) {
        events.setText("Long click: " + v.toString());
        return true;
    }
});
```

First, the `Button` object is requested by providing its identifier. Then the `setOnLongClickListener()` method is called with our implementation of the `View.OnLongClickListener` class interface. The `View` on which the user long-clicked is passed in to the `onLongClick()` event handler. Here again we use the same `TextView` as before to display text saying that a long click occurred.

Listening for Focus Changes

We have already discussed listening for focus changes on an entire screen. All `View` objects, though, can also trigger a call to listeners when their particular focus state changes. You do this by providing an implementation of the `View.OnFocusChangeListener` class to the `setOnFocusChangeListener()` method. The following is an example of how to listen for focus change events with an `EditText` control:

```
TextView focus = (TextView) findViewById(R.id.text_focus_change);
focus.setOnFocusChangeListener(new View.OnFocusChangeListener() {
    public void onFocusChange(View v, boolean hasFocus) {
        if (hasFocus) {
            if (mSaveText != null) {
                ((TextView)v).setText(mSaveText);
            }
        } else {
            mSaveText = ((TextView)v).getText().toString();
            ((TextView)v).setText("");
        }
    }
});
```

In this implementation, we also use a private member variable of type `String` for `mSaveText`. After retrieving the `EditText` control as a `TextView`, we do one of two things. If the user moves focus away from the control, we store the text in `mSaveText` and set the text to empty. If the user changes focus to the control, though, we restore this text. This has the amusing effect of hiding the text the user entered when the control is not active. This can be useful on a form on which a user needs to make multiple, lengthy

text entries but you want to provide an easy way for the user to see which one to edit. It is also useful for demonstrating a purpose for the focus listeners on a text entry. Other uses might include validating text a user enters after the user navigates away or prefilling the text entry the first time the user navigates to it with something else entered.

Working with Gestures

Android devices often rely on touchscreens for user input. Users are now quite comfortable using common finger gestures to operate their devices. Android applications can detect and react to one-finger (single-touch) and two-finger (multitouch) gestures. Users can also use gestures with the drag-and-drop framework to enable the arrangement of `View` controls on a device screen.



Note

Even early Android devices supported simple single-touch gestures. Support for multitouch gestures was added in the Android 2.2 SDK and is available only on devices with capacitive touchscreen hardware. Some capacitive hardware is capable of tracking up to ten different points at once.

One of the reasons that gestures can be a bit tricky is that a gesture can be made of multiple touch events or motions. Different sequences of motion add up to different gestures. For example, a fling gesture involves the user pressing a finger down on the screen, swiping across the screen, and lifting the finger up off the screen while the swipe is still in motion (that is, without slowing down to stop before lifting the finger). Each of these steps can trigger motion events to which applications can react.

Detecting User Motions within a View

By now you've come to understand that Android application user interfaces are built using different types of `View` controls. Developers can handle gestures much as they do click events within a `View` control using the `setOnClickListener()` and `setOnLongClickListener()` methods. Instead, the `onTouchEvent()` callback method is used to detect that some motion has occurred within the `View` region.

The `onTouchEvent()` callback method has a single parameter, a `MotionEvent` object. The `MotionEvent` object contains all sorts of details about what kind of motion occurs in the `View`, enabling the developer to determine what sort of gesture is happening by collecting and analyzing many consecutive `MotionEvent` objects. You can use all of the `MotionEvent` data to recognize and detect every kind of gesture you can possibly imagine. Alternatively, you can use built-in gesture detectors provided in the Android SDK to detect common user motions in a consistent fashion. Android currently has two different classes that can detect navigational gestures:

- The `GestureDetector` class can be used to detect common single-touch gestures.
- The `ScaleGestureDetector` can be used to detect multitouch scale gestures.

It is likely that more gesture detectors will be added in future versions of the Android SDK. You can also implement your own gesture detectors to detect any gestures not supported by the built-in ones. For example, you might want to create a two-fingered rotate gesture to, say, rotate an image, or a three-fingered swipe gesture that brings up an options menu.

In addition to common navigational gestures, you can use the `android.gesture` package with the `GestureOverlayView` to recognize commandlike gestures. For instance, you can create an S-shaped gesture that brings up a search or a zigzag gesture that clears the screen on a drawing app. Tools are available for recording and creating libraries of this style of gesture. As it uses an overlay for detection, it isn't well suited for all types of applications. This package was introduced in API Level 4.



Warning

- The type and sensitivity of the touchscreen can vary by device. Different devices can detect different numbers of touch points simultaneously, which affects the complexity of gestures you can support.

Handling Common Single-Touch Gestures

Introduced in API Level 1, the `GestureDetector` class can be used to detect gestures made by a single finger. Some common single-finger gestures supported by the `GestureDetector` class include:

- `onDown`: Called when the user first presses the touchscreen.
- `onShowPress`: Called after the user first presses the touchscreen but before lifting the finger or moving it around on the screen; used to visually or audibly indicate that the press has been detected.
- `onSingleTapUp`: Called when the user lifts up (using the up `MotionEvent`) from the touchscreen as part of a single-tap event.
- `onSingleTapConfirmed`: Called when a single-tap event occurs.
- `onDoubleTap`: Called when a double-tap event occurs.
- `onDoubleTapEvent`: Called when an event within a double-tap gesture occurs, including any down, move, or up `MotionEvent`.
- `onLongPress`: Similar to `onSingleTapUp`, but called if the user holds down a finger long enough to not be a standard click but also without any movement.
- `onScroll`: Called after the user presses and then moves a finger in a steady motion before lifting the finger. This is commonly called *dragging*.
- `onFling`: Called after the user presses and then moves a finger in an accelerating motion before lifting it. This is commonly called a *flick gesture* and usually results in some motion continuing after the user lifts the finger.

You can use the interfaces available with the `GestureDetector` class to listen for specific gestures such as single and double taps (see `GestureDetector.OnDoubleTapListener`), as well as scrolls and flings (see the documentation for `GestureDetector.OnGestureListener`). The scrolling gesture involves touching the screen and moving a finger around on it. The fling gesture, on the other hand, causes (though not automatically) the object to continue to move even after the finger has been lifted from the screen. This gives the user the impression of throwing or flicking the object around on the screen.



Tip

- You can use the `GestureDetector.SimpleOnGestureListener` class to listen to any and all of the gestures recognized by the `GestureDetector`.

Let's look at a simple example. Let's assume you have a game screen that enables the user to perform gestures to interact with a graphic on the screen. We can create a custom `View` class called `GameAreaView` that can dictate how a bitmap graphic moves around within the game area based upon each gesture. The `GameAreaView` class can use the `onTouchEvent()` method to pass along `MotionEvent` objects to a `GestureDetector`. In this way, the `GameAreaView` can react to simple gestures, interpret them, and make the appropriate changes to the bitmap, including moving it from one location to another on the screen.



Tip

- How the gestures are interpreted and what actions they cause are completely up to the developer. You can, for example, interpret a fling gesture and make the bitmap graphic disappear . . . but does that make sense? Not really. It's important to always make the gesture jibe well with the resulting operation in the application so that users are not confused. Users are now accustomed to specific screen behavior based on certain gestures, so it's best to use the expected convention, too.

In this case, the `GameAreaView` class interprets gestures as follows:

- A double-tap gesture causes the bitmap graphic to return to its initial position.
- A scroll gesture causes the bitmap graphic to “follow” the motion of the finger.
- A fling gesture causes the bitmap graphic to “fly” in the direction of the fling.



Tip

- Many of the code examples provided in this section are taken from the `SimpleGestures` application. The source code for this application is provided for download on the book's website.

To make these gestures work, the `GameAreaView` class needs to include the appropriate gesture detector, which triggers any operations upon the bitmap graphic. Based upon the specific gestures detected, the `GameAreaView` class must perform all translation animations and other graphical operations applied to the bitmap. To wire up the

GameAreaView class for gesture support, we need to implement several important methods:

- The class constructor must initialize any gesture detectors and bitmap graphics.
- The `onTouchEvent()` method must be overridden to pass the `MotionEvent` data to the gesture detector for processing.
- The `onDraw()` method must be overridden to draw the bitmap graphic in the appropriate position at any time.
- Various methods are needed to perform the graphics operations required to make a bitmap move around on the screen, fly across the screen, and reset its location based upon the data provided by the specific gesture.

All these tasks are handled by our `GameAreaView` class definition:

```
public class GameAreaView extends View {
    private static final String DEBUG_TAG =
        "SimpleGestures->GameAreaView";
    private GestureDetector gestures;
    private Matrix translate;
    private Bitmap droid;
    private Matrix animateStart;
    private Interpolator animateInterpolator;
    private long startTime;
    private long endTime;
    private float totalAnimDx;
    private float totalAnimDy;

    public GameAreaView(Context context, int iGraphicResourceId) {
        super(context);
        translate = new Matrix();
        GestureListener listener = new GestureListener(this);
        gestures = new GestureDetector(context, listener, null, true);
        droid = BitmapFactory.decodeResource(getResources(),
            iGraphicResourceId);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        boolean retVal = false;
        retVal = gestures.onTouchEvent(event);
        return retVal;
    }

    @Override
    protected void onDraw(Canvas canvas) {
        Log.v(DEBUG_TAG, "onDraw");
        canvas.drawBitmap(droid, translate, null);
    }
}
```

```

public void onResetLocation() {
    translate.reset();
    invalidate();
}

public void onMove(float dx, float dy) {
    translate.postTranslate(dx, dy);
    invalidate();
}

public void onAnimateMove(float dx, float dy, long duration) {
    animateStart = new Matrix(translate);
    animateInterpolator = new OvershootInterpolator();
    startTime = android.os.SystemClock.elapsedRealtime();
    endTime = startTime + duration;
    totalAnimDx = dx;
    totalAnimDy = dy;
    post(new Runnable() {
        @Override
        public void run() {
            onAnimateStep();
        }
    });
}

private void onAnimateStep() {
    long curTime = android.os.SystemClock.elapsedRealtime();
    float percentTime = (float) (curTime - startTime) /
        (float) (endTime - startTime);
    float percentDistance = animateInterpolator
        .getInterpolation(percentTime);
    float curDx = percentDistance * totalAnimDx;
    float curDy = percentDistance * totalAnimDy;
    translate.set(animateStart);
    onMove(curDx, curDy);

    if (percentTime < 1.0f) {
        post(new Runnable() {
            @Override
            public void run() {
                onAnimateStep();
            }
        });
    }
}
}

```

As you can see, the `GameAreaView` class keeps track of where the bitmap graphic should be drawn at any time. The `onTouchEvent()` method is used to capture motion events and pass them along to a gesture detector whose `GestureListener` we must implement as well (more on this in a moment). Typically, each method of the `GameAreaView` applies some operation to the bitmap graphic and then calls the `invalidate()` method, forcing the `View` to be redrawn.

Now we turn our attention to the methods required to implement specific gestures:

- For double-tap gestures, we implement a method called `onResetLocation()` to draw the bitmap graphic in its original location.
- For scroll gestures, we implement a method called `onMove()` to draw the bitmap graphic in a new location. Note that scrolling can occur in any direction—it simply refers to a finger swipe on the screen.
- For fling gestures, things get a little tricky. To animate motion on the screen smoothly, we used a chain of asynchronous calls and a built-in Android interpolator to calculate the location in which to draw the graphic based upon how long it has been since the animation started. See the `onAnimateMove()` and `onAnimateStep()` methods for the full implementation of fling animation.

Now we need to implement our `GestureListener` class to interpret the appropriate gestures and call the `GameAreaView` methods we just implemented. Here's an implementation of the `GestureListener` class that our `GameAreaView` class can use:

```
private class GestureListener extends
    GestureDetector.SimpleOnGestureListener {

    GameAreaView view;

    public GestureListener(GameAreaView view) {
        this.view = view;
    }

    @Override
    public boolean onDown(MotionEvent e) {
        return true;
    }

    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2,
        final float velocityX, final float velocityY) {
        final float distanceTimeFactor = 0.4f;
        final float totalDx = (distanceTimeFactor * velocityX / 2);
        final float totalDy = (distanceTimeFactor * velocityY / 2);

        view.onAnimateMove(totalDx, totalDy,
            (long) (1000 * distanceTimeFactor));
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent e) {
        view.onResetLocation();
        return true;
    }
}
```

```

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2,
    float distanceX, float distanceY) {
    view.onMove(-distanceX, -distanceY);
    return true;
}
}

```

Note that you must return `true` for any gesture or motion event that you want to detect. Therefore, you must return `true` in the `onDown()` method as it happens at the beginning of a scroll-type gesture. Most of the implementation of the `GestureListener` class methods involves our interpretation of the data for each gesture. For example:

- We react to double taps by resetting the bitmap to its original location using the `onResetLocation()` method of our `GameAreaView` class.
- We use the distance data provided in the `onScroll()` method to determine the direction to use in the movement to pass in to the `onMove()` method of the `GameAreaView` class.
- We use the velocity data provided in the `onFling()` method to determine the direction and speed to use in the movement animation of the bitmap. The `timeDistanceFactor` variable with a value of 0.4 is subjective; it gives the resulting slide-to-a-stop animation enough time to be visible but is short enough to be controllable and responsive. You can think of it as a high-friction surface. This information is used by the animation sequence implemented in the `onAnimateMove()` method of the `GameAreaView` class.

Now that we have implemented the `GameAreaView` class in its entirety, you can display it on a screen. For example, you might create an `Activity` that has a user interface with a `FrameLayout` control and add an instance of a `GameAreaView` using the `addView()` method. Figure 8.3 shows a gesture example of dragging a droid around a screen.



Tip

To support the broadest range of devices, we recommend supporting simple, one-fingered gestures and providing alternative navigational items for devices that don't support multitouch gestures. However, users are beginning to expect support for multitouch gestures now, so use them where you can and where they make sense. Resistive touchscreens remain uncommon, typically appearing only on lower-end devices.

Handling Common Multitouch Gestures

Introduced in API Level 8 (Android 2.2), the `ScaleGestureDetector` class can be used to detect two-fingered scale gestures. The scale gesture enables the user to move two

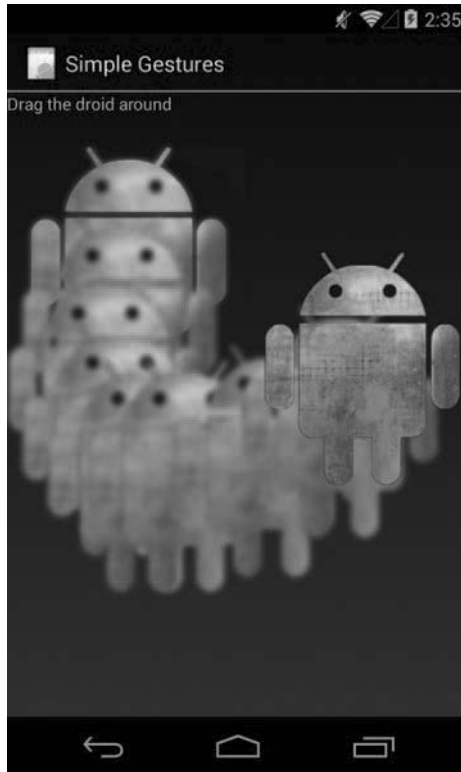


Figure 8.3 Using gestures to drag the droid around the screen.

fingers toward and away from each other. Moving the fingers apart is considered scaling up; moving the fingers together is considered scaling down. This is the “pinch-to-zoom” style often employed by map and photo applications.



Tip

- You can use the `ScaleGestureDetector.SimpleOnScaleGestureListener` class to detect scale gestures detected by the `ScaleGestureDetector`.

Let’s look at another example. Again, we use the custom `View` class called `GameAreaView`, but this time we handle the multitouch scale event. In this way, the `GameAreaView` can react to scale gestures, interpret them, and make the appropriate changes to the bitmap, including growing or shrinking it on the screen.



Tip

- Many of the code examples provided in this section are taken from the `SimpleMultiTouchGesture` application. The source code for this application is provided for download on the book’s website.

To handle scale gestures, the `GameAreaView` class needs to include the appropriate gesture detector, a `ScaleGestureDetector`. The `GameAreaView` class needs to be wired up for scale gesture support similarly to the single-touch gestures implemented earlier, including initializing the gesture detector in the class constructor, overriding the `onTouchEvent()` method to pass the `MotionEvent` objects to the gesture detector, and overriding the `onDraw()` method to draw the `View` appropriately as necessary. We also need to update the `GameAreaView` class to keep track of the bitmap graphic size (using a `Matrix`) and provide a helper method for growing or shrinking the graphic. Here is the new implementation of the `GameAreaView` class with scale gesture support:

```
public class GameAreaView extends View {
    private ScaleGestureDetector multiGestures;
    private Matrix scale;
    private Bitmap droid;

    public GameAreaView(Context context, int iGraphicResourceId) {
        super(context);
        scale = new Matrix();
        GestureListener listener = new GestureListener(this);
        multiGestures = new ScaleGestureDetector(context, listener);
        droid = BitmapFactory.decodeResource(getResources(),
            iGraphicResourceId);
    }

    public void onScale(float factor) {
        scale.preScale(factor, factor);
        invalidate();
    }

    @Override
    protected void onDraw(Canvas canvas) {
        Matrix transform = new Matrix(scale);
        float width = droid.getWidth() / 2;
        float height = droid.getHeight() / 2;
        transform.postTranslate(-width, -height);
        transform.postConcat(scale);
        transform.postTranslate(width, height);
        canvas.drawBitmap(droid, transform, null);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        boolean retVal = false;
        retVal = multiGestures.onTouchEvent(event);
        return retVal;
    }
}
```

As you can see, the `GameAreaView` class keeps track of what size the bitmap should be at any time using the `Matrix` variable called `scale`. The `onTouchEvent()` method is used to capture motion events and pass them along to a `ScaleGestureDetector` gesture detector. As before, the `onScale()` helper method of the `GameAreaView` applies some scaling to the bitmap graphic and then calls the `invalidate()` method, forcing the `View` to be redrawn.

Now let's take a look at the `GestureListener` class implementation necessary to interpret the scale gestures and call the `GameAreaView` methods we just implemented. Here's the implementation of the `GestureListener` class:

```
private class GestureListener implements
    ScaleGestureDetector.OnScaleGestureListener {

    GameAreaView view;

    public GestureListener(GameAreaView view) {
        this.view = view;
    }

    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        float scale = detector.getScaleFactor();
        view.onScale(scale);
        return true;
    }

    @Override
    public boolean onScaleBegin(ScaleGestureDetector detector) {
        return true;
    }

    @Override
    public void onScaleEnd(ScaleGestureDetector detector) {
    }
}
```

Remember that you must return `true` for any gesture or motion event that you want to detect. Therefore, you must return `true` in the `onScaleBegin()` method as it happens at the beginning of a scale-type gesture. Most of the implementation of the `GestureListener` methods involves our interpretation of the data for the scale gesture. Specifically, we use the scale factor (provided by the `getScaleFactor()` method) to calculate whether we should shrink or grow the bitmap graphic, and by how much. We pass this information to the `onScale()` helper method we just implemented in the `GameAreaView` class.

Now, if you were to use the `GameAreaView` class in your application, scale gestures might look something like Figure 8.4.



Note

The Android emulator does not currently support multitouch input, although there is experimental support in the works. You will have to run and test multitouch input such as the scale gesture using a device running Android 2.2 or higher.

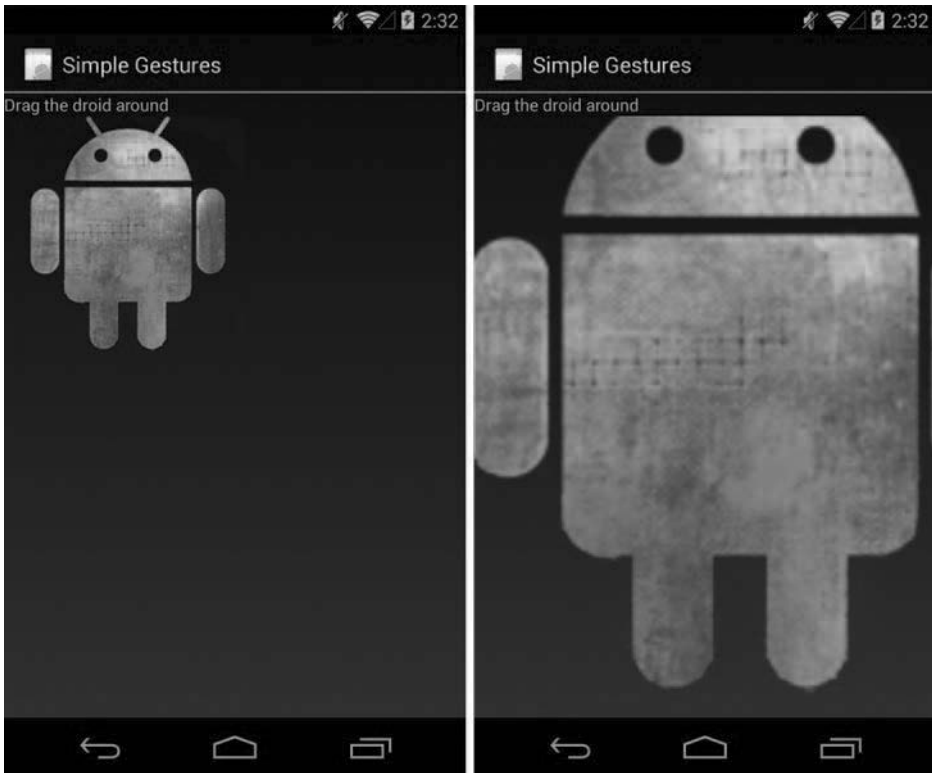


Figure 8.4 The result of scale-down (left) and scale-up (right) gestures.

Making Gestures Look Natural

Gestures can enhance your Android application user interfaces in new, interesting, and intuitive ways. Closely mapping the operations being performed on the screen to the user's finger motion makes a gesture feel natural and intuitive. Making application operations look natural requires some experimentation on the part of the developer. Keep in mind that devices vary in processing power, and this might be a factor in making things seem natural. Minimal processing, even on fast devices, will help keep gestures and the reaction to them smooth and responsive, and thus natural-feeling.

Using the Drag-and-Drop Framework

On Android devices running Android 3.0 and higher (API Level 11), developers can access the drag-and-drop framework to perform drag-and-drop actions. You can drag and drop `View` controls within the scope of a screen or `Activity` class.

The drag-and-drop process basically works like this:

- The user triggers a drag operation. How this is done depends on the application, but long clicks are a reasonable option for selecting a `View` for a drag under the appropriate conditions.
- The data for the selected `View` control is packaged in a `ClipData` object (also used by the clipboard framework), and the `View.DragShadowBuilder` class is used to generate a little visual representation of the item being dragged. For example, if you were dragging a filename into a directory bucket, you might include a little icon of a file.
- You call the `startDrag()` method on the `View` control to be dragged. This starts a drag event. The system signals a drag event with `ACTION_DRAG_STARTED`, which listeners can catch.
- There are a number of events that occur during a drag that your application can react to. The `ACTION_DRAG_ENTERED` event can be used to adjust the screen controls to highlight other `View` controls that the dragged `View` control might want to be dragged over to. The `ACTION_DRAG_LOCATION` event can be used to determine where the dragged `View` is on the screen. The `ACTION_DRAG_EXITED` event can be used to reset any screen controls that were adjusted in the `ACTION_DRAG_ENTERED` event.
- When the user ends the drag operation by releasing the shadow item over a specific target `View` on the screen, the system signals a drop event with `ACTION_DROP`, which listeners can catch. Any data can be retrieved using the `getClipData()` method.

For more information about the drag-and-drop framework, see the Android SDK documentation. There you can also find a great example of using the drag-and-drop framework called `DragAndDropDemo.java`.

Handling Screen Orientation Changes

Android devices have both landscape and portrait modes and can seamlessly transition between these orientations. The Android operating system automatically handles these changes for your application, if you so choose. You can also provide alternative resources, such as different layouts, for portrait and landscape modes. Also, you can directly access device sensors such as the accelerometer, which we talk about in Chapter 15, “Accessing Android’s Hardware Sensors,” to capture device orientation along three axes.

However, if you want to listen for simple screen orientation changes programmatically and have your application react to them, you can use the `OrientationEventListener` class to do this within your `Activity`.



Tip

- Many of the code examples provided in this section are taken from the `SimpleOrientation` application. The source code for this application is provided for download on the book's website.



Warning

- Orientation changes are best tested on real devices, not with the emulator.

Implementing orientation event handling in your `Activity` is simple. Simply instantiate an `OrientationEventListener` and provide its implementation. For example, the following `Activity` class called `SimpleOrientationActivity` logs orientation information to `LogCat`:

```
public class SimpleOrientationActivity extends Activity {
    OrientationEventListener mOrientationListener;

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

        mOrientationListener = new OrientationEventListener(this,
            SensorManager.SENSOR_DELAY_NORMAL) {

            @Override
            public void onOrientationChanged(int orientation) {
                Log.v(DEBUG_TAG,
                    "Orientation changed to " + orientation);
            }
        };

        if (mOrientationListener.canDetectOrientation() == true) {
            Log.v(DEBUG_TAG, "Can detect orientation");
            mOrientationListener.enable();
        } else {
            Log.v(DEBUG_TAG, "Cannot detect orientation");
            mOrientationListener.disable();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mOrientationListener.disable();
    }
}
```

You can set the rate to check for orientation changes to a variety of different values. There are other rate values appropriate for game use and other purposes. The default rate,

`SENSOR_DELAY_NORMAL`, is most appropriate for simple orientation changes. Other values, such as `SENSOR_DELAY_UI` and `SENSOR_DELAY_GAME`, might make sense for your application.

After you have a valid `OrientationEventListener` object, you can check if it can detect orientation changes using the `canDetectOrientation()` method, and enable and disable the listener using its `enable()` and `disable()` methods.

The `OrientationEventListener` has a single callback method, which enables you to listen for orientation transitions, the `onOrientationChanged()` method. This method has a single parameter, an `integer`. This integer normally represents the device tilt as a number between 0 and 359:

- A result of `ORIENTATION_UNKNOWN` (-1) means the device is flat (perhaps on a table) and the orientation is unknown.
- A result of 0 means the device is in its “normal” orientation, with the top of the device facing in the up direction. (“Normal” is defined by the device manufacturer. You need to test on each device to find out for sure what “normal” means.)
- A result of 90 means the device is tilted 90 degrees, with the left side of the device facing in the up direction.
- A result of 180 means the device is tilted 180 degrees, with the bottom side of the device facing in the up direction (upside down).
- A result of 270 means the device is tilted 270 degrees, with the right side of the device facing in the up direction.

Figure 8.5 shows an example of how the device orientation might read when the device is tilted to the right by 91 degrees.

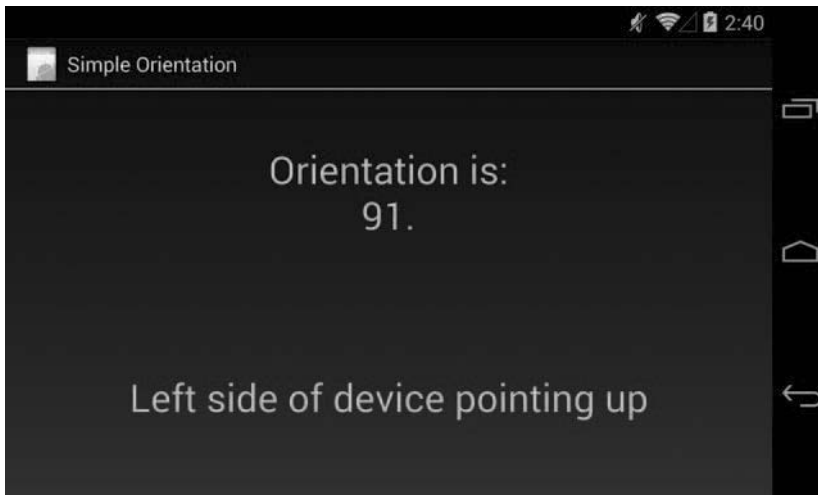


Figure 8.5 Orientation of the device as reported by an `OrientationEventListener`.

Summary

The Android platform enables great flexibility when it comes to ways that users can provide input to the device. Developers benefit from the fact that many powerful input methods are built into the `View` controls themselves, just waiting to be leveraged. Applications can take advantage of built-in input methods, such as software keyboards, or can customize them for special purposes. The Android framework also includes powerful features, such as a clipboard `Service`, gesture support, and a drag-and-drop framework, that your applications can use. It is important to support a variety of input methods in your applications, as users often have distinct preferences and not all methods are available on all devices.

Quiz Questions

1. True or false: IME stands for Input Method Editor.
2. Name the observer discussed in this chapter that listens to a `View` and all its child `View` objects.
3. What are two classes that are able to detect navigational gestures?
4. What method is called for a dragging single-finger gesture?
5. True or false: The `MultiGestureDetector` class can be used to detect two-fingered scale gestures.

Exercises

1. Use the online documentation to create a list of the core gestures supported by Android.
2. Modify the `SimpleGestures` application so that it makes use of the double-touch drag gesture design pattern.
3. Use the online documentation to create a list of the different `inputType` constants and their associated constant values.

References and More Information

Android API Guides: “Copy and Paste”:

<http://d.android.com/guide/topics/text/copy-paste.html>

Android SDK Reference regarding the `ClipboardManager`:

<http://d.android.com/reference/android/content/ClipboardManager.html>

Android SDK Reference regarding the `ClipData` class:

<http://d.android.com/reference/android/content/ClipData.html>

Android API Guides: “Drag and Drop”:

<http://d.android.com/guide/topics/ui/drag-drop.html>

Android SDK Reference regarding the `android.gesture` package:

<http://d.android.com/reference/android/gesture/package-summary.html>

Android Design: “Gestures”:

<http://d.android.com/design/patterns/gestures.html>

Index

Symbols

- * (asterisk), filtering log information, 475
- . (dot), sqlite3 commands, 486
- ; (semicolon), sqlite3, 490

A

- AbstractAccountAuthenticator class, 428**
- AbstractThreadedSyncAdapter class, 429**
- AccelerateDecelerateInterpolator, 341**
- AccelerateInterpolator, 341**
- ACCESS_COARSE_LOCATION permission, 212**
- Access control, SQLite limitations, 491**
- Access points, listing, 248**
- Accessible applications**
 - framework, 139–141
 - overview of, 139
 - quiz Q & A, 148, 520
 - speech recognition services, 141–145
 - testing, 147
 - text-to-speech services, 145–147
- Accessories**
 - new Android hardware, 239–240
 - USB, 240–242
- Account authenticator, 428**
- Account provider, 428**
- AccountManager class, 427–429**
- Accounts. See User accounts**
- ACCURACY_COARSE, location services, 255**

ACCURACY_COARSE_LOCATION, 261

ACCURACY_FINE, location services, 255

ACCURACY_FINE_LOCATION, 261

Achievements, Google Play game services, 297–298

Action bars

- building basic, 98–101
- contextual action mode, 105–106
- customizing, 101–103
- handling application icon clicks on, 103–104
- overview of, 98–99
- working with screens not requiring, 104–105

ActionMode class, 104

ACTION_RECOGNIZE_SPEECH, 145

ACTION_REQUEST_DISCOVERABLE intent, Bluetooth, 237

ACTION_REQUEST_ENABLE intent, Bluetooth, 237

ActionScript 3, 188

ACTION_SEARCH, 422–423

ACTION_VIEW intent, Google Maps, 263

Activity class

- application acting as content filter, 410–412
- asynchronously loading data, 16
- AsyncTask, 13–14
- building action bars, 98–101
- configuring default messaging, 217
- creating App Widget, 393
- creating search, 422–423
- data for Google Analytics, 287
- gathering statistics, 292
- GLSurfaceView, 366–369
- IntentService class, 31–33
- launching browser, 176
- native activities, 384

- OpenGL/application threads, 362–364

- removing action bars, 104–105

- software keyboards, 116

- themes, 111–113

- Thread class, 15–16

Activity launch, 341, 422

Activity lifecycle

- AsyncTask class, 12–14
- spanning processing across, 19
- text notifications, 80

ActivityOptions class, 341

Activity recognition APIs, Google location services, 261–262

<activity> tag

- application acting as content filter, 411
- enabling application search, 424
- hardware acceleration control, 325
- registering intent filter, 413
- themes, 111

ADB (Android Debug Bridge)

- accessing sqlite3 from, 486
- backup service controls, 476–477
- copying files, 472
- directing commands to specific devices, 470
- generating bug reports, 477–478
- inspecting SQLite databases with shell, 478
- installing custom binaries, 481–482
- installing/uninstalling applications, 473
- issuing shell commands, 471–472
- listing all commands, 470–471
- listing connected devices/emulators, 469
- with LogCat logging, 474–476
- overview of, 469
- quiz Q & A, 482, 526

- starting ADB server, 470
- stopping ADB server, 470
- stress testing applications with shell, 478–481
- addGlobalLayoutListener() method, 121**
- addHelper() method, 432–434**
- addJavascriptInterface() method, 183, 187**
- addOnDrawListener() method, 121**
- addOnPreDrawListener() method, 120**
- addOnTouchModeChangeListener() method, 120**
- addURI() method, 419**
- ADK (Android Accessory Development Kit), 139, 239–240**
- Admin permissions, Bluetooth, 235, 237–238**
- Admin section, Google Analytics, 284–285**
- Adobe Air, 187–188**
- Adobe Flash, 187–188**
- ADT (Android Development Tools), 111, 500**
- ADT-1 Developer Kit, Android TV, 465**
- Ahead-of-time (AOT) compilation, ART runtime, 460**
- AIDL (Android Interface Definition Language), 26–28**
- AIR, Adobe, 187–188**
- AlarmManager class, App Widgets, 396**
- Alarms, system event, 208**
- Alerts, proximity, 260**
- All Apps button, language settings, 442**
- Alpha transparency transformations, 334–335**
- ALTER TABLE, SQLite limitations, 491**
- Alternative resources**
 - changing language settings, 442–444
 - device diversity, 153
 - internationalization, 439–442
- Amazon Appstore for Android, 280**
- Android 4.4 (KitKat), 6, 508**
- Android Accessory Development Kit (ADK), 139, 239–240**
- Android Backup Service, 430–435**
- Android Beam**
 - configuring manifest file, 244–245
 - enabling sending, 241–243
 - host card emulation, 245
 - over Bluetooth, 245
 - overview of, 241
 - receiving messages, 243–244
- Android Debug Bridge. See ADB (Android Debug Bridge)**
- Android Developers Blog, 117**
- Android Development Tools (ADT), 111, 500**
- Android IDE, 499–500**
- Android Interface Definition Language (AIDL), 26–28**
- Android location APIs**
 - doing more, 260
 - geocoding locations, 256–260
 - GPS, 254–256
 - overview of, 253
- Android NDK (Native Development Kit)**
 - drawbacks, 377–378
 - improving graphics performance, 384–385
 - installing, 378
 - leveraging OpenGL ES, 346
 - quiz Q & A, 386, 524
 - RenderScript vs., 385–386
 - sample application, 379
 - when to use, 377–378
- Android NDK (Native Development Kit), creating project**
 - calling native code from Java, 380–381
 - overview of, 379–380

Android NDK (Native Development Kit),**creating project** (*continued*)

- parameters and return values, 381–382

- using exceptions with native code, 382–383

- using native activities, 384

Android Runtime (ART), L Developer Preview, 460–461**Android SDK**

- configuring Android Studio, 508–509

- getting familiar with Java documentation, 500

- License Agreement, 174

- OpenGL ES APIs in, 347

- OpenGL ES in Android, 346

- using Android NDK vs., 377–378

- versions of OpenGL ES, 346

Android Studio

- configuring, 508–509

- creating project, 509–512

- debugging applications, 515–517

- getting up and running, 507–508

- Gradle build system, 513

- keyboard shortcuts, 517

- launching for first time, 508

- Layout Editor, 513–515

- learning Java development tools, 500

- overview of, 507

- project structure, 512

- quiz Q & A, 517–518, 526

- user interface, 513–514

Android TV, 464–465**Android Wear API, 158–159****android.accounts package, 427–429****android.animation package, 339****android.database.sqlite package, 36****AndroidManifest.xml file**

- building content provider, 62

- configuring Android Beam, 244–245

- configuring App Widgets, 399

- configuring live wallpapers, 406–407

- configuring search, 423–424

- creating Service, 20

- enabling vibration with notifications, 84

- permissions, 25

- registering backup agent, 434

- securing application broadcasts, 74

- working with themes, 111–113

android.permission.INTERNET permission, 175–176**android.transition, animation, 342****android.view.animation package**

- alpha transparency transformations, 335

- Interpolator class, 341

- loading animations, 334

- moving transformations, 336

- rotating transformations, 335

- scaling transformations, 336

- tweened animations, 333

android.webkit package, 182–187**animate() method, property animations, 339–340****animateCamera() method, 267****Animation**

- Activity launch, 341

- drawable, 329–331

- GIF images, 329–331

- in L Developer Preview, 462–463

- property, 336–341

- quiz Q & A, 342–343, 524

- scenes and transitions for state, 342

- types of graphic, 329

- view, 331–336
- working with interpolators, 341
- AnimationDrawable class, 330**
- AnimationListener class, 334**
- AnimationSet, 333**
- AnimationUtils helper class, 334**
- Animator.AnimatorPauseListener, property animation, 337**
- ANR (Application Not Responding) events, 11–12**
- Anti-aliasing, Paint, 307**
- AnticipateInterpolator, 341**
- AnticipateOvershootInterpolator, 341**
- Antipiracy. See also Software piracy protection, 299–300**
- AOT (ahead-of-time) compilation, ART runtime, 460**
- API Access link, maps, 263**
- APIs, Android**
 - Android Wear, 158–159
 - multimedia. *See* **Multimedia APIs**
 - networking. *See* **Networking APIs**
 - optional hardware. *See* **Hardware APIs**
 - telephony. *See* **Telephony APIs**
 - web APIs. *See* **Web APIs**
- Apple, Siri speech-recognizing assistant, 139**
- Application Context, SQLite database, 36, 46**
- Application lifecycle, 41**
- Application Not Responding (ANR) events, 11–12**
- Applications**
 - App Widgets tied to underlying, 392
 - content providers for, 62–65
 - gathering statistics. *See* **Google Analytics**
 - gathering statistics from, 292
 - handling icon clicks on action bar, 103–104
 - searching. *See* **Search**
- <application> tag, AndroidManifest.xml file**
 - controlling hardware acceleration, 325
 - registering backup agent, 434
 - restricted profiles, 429
 - right-to-left language localization, 445
- AppStateManager class, 299**
- AppWidgetProvider class, 395–397**
- <appwidget-provider> tag, 394, 402**
- App Widgets**
 - adding to Lock screen, 401–403
 - becoming host, 401
 - calculating size of, 394
 - configuring Android manifest file for, 399
 - creating, 393–396
 - defined, 392
 - installing to Home screen, 400–401
 - updating, 397–399
 - using remote views, 396–397
 - working with, 392–393
- Archived data, wiping, 477**
- Arcs, drawing, 320–322**
- ArgbEvaluator class, property animation, 337**
- ARMv5TE devices, Android NDK, 379–380**
- ArrayAdapter, binding data to controls, 53**
- Arrays, drawing vertices, 353–355**
- ART (Android Runtime), L Developer Preview, 460–461**
- Asterisk (*), filtering log information, 475**
- Asynchronous network operations, 167–169, 260**
- AsyncTask class, 12–14, 168, 314**
- Attributes, property animation, 338**

Audiences

- attracting new types of device, 153
- for this book, 1

Audio

- playing, 205–206
- recording, 204–205
- ringtones, 208–209
- sharing, 206–207

AudioManager, 206**Auditing, SQLite limitations, 491****Authors of this book, contacting, 8****AutoCompleteTextView control, 50, 53, 118****AUTOINCREMENT, SQLite database tables, 492****AVD, Android location services APIs, 256**

B

Background processing

- AsyncTask, 12–14
- Service, 19–20, 22–24

Backup agent, 431–434**Backup helper, 431–432****Backup Manager, 434–435, 476–477****Backup service**

- choosing remote, 430–431
- controlling using ADB command, 476–477
- forcing backup operations, 477
- forcing restore operations, 477
- implementing backup agent, 432–435
- overview of, 430
- registering with Android, 432
- requesting backup, 434
- requesting restore, 435
- wiping archived data, 477

BaseColumns interface, database field names, 47**BaseGameActivity class, Google Play game services, 296–297****BasicGLThread class, OpenGL ES, 349–350****basicNativeCall() method, 381****Battery**

- monitoring use of, 231–233
- optimization in Project Volta, 461

beginTransaction() method, SQLite database transactions, 40**Bidi Formatter utility class, 445****BigPictureStyle class, notifications, 88–90****bigText() method, notifications, 88–90****BigTextStyle class, notifications, 88–90****Binding data, to application user interface, 48–53****bindService() method, 20–21, 27****Bitmap graphics**

- 2D applications, 312–315
- buffering issues of large, 170
- drawable animation, 330–331
- drawing on Canvas, 313
- performance optimizations, 314–315
- scaling, 313
- texturing 3D objects, 361–362
- transforming using Matrix, 313–314
- working with, 312
- working with view animations, 331–332

BLE (Bluetooth Low Energy) peripheral devices, 235–236**Blinking lights, notifications, 84–85****Blocking operations, 11****Bluetooth**

- Android Beam over, 245
- Android support for, 235–236
- checking hardware for, 236–237
- discovering devices, 237–238
- enabling, 237

- establishing connections between devices, 238–239
- querying for paired devices, 237
- quiz Q & A, 248, 522
- BluetoothAdapter class, 236–238**
- BLUETOOTH_ADMIN permission, 235, 237–238**
- Bluetooth Low Energy (BLE) peripheral devices, 235–236**
- BLUETOOTH permission, 235**
- BluetoothSocket object, 238**
- Body text, notifications, 79**
- BounceInterpolator, animation, 341**
- Breakpoints, debugging Android Studio applications, 515–516**
- BroadcastIntent class, 31–32, 207, 231**
- BroadcastReceiver class**
 - default messaging application, 217
 - GSM message flow, 272
 - monitoring device battery, 231–233
 - monitoring Wi-Fi state, 247
 - receiving broadcasts, 69–71
- Broadcasts**
 - overview of, 67
 - quiz Q & A, 74, 520
 - receiving, 69–74
 - securing application, 73–74
 - sending, 67–68
- Browser chrome, WebView control, 179–180**
- Buffering issues, large bitmaps, 170**
- Buffers**
 - coloring vertices, 355–356
 - converting arrays to, 354–355
 - drawing complex 3D, 356–358
 - texture coordinate, 361–362
- build() method**
 - animating map camera, 267
 - creating text notification with icon, 80

- Builder classes, 290–291, 501**
- Built-in functions, SQLite limitations, 491**
- Button control**
 - applying styles, 108
 - IntentService, 31–33
 - listening for long click on, 122
 - recording speech, 145
 - web extensions, 183, 185
- ByteBuffer, drawing 3D objects, 356–358**

C

- C2DM (Cloud to Device Messaging), 271**
- CacheManager class, WebKit API, 182**
- calculateAndDisplayFPS() method, OpenGL, 362–364**
- Calculated columns, SQLite database, 496–497**
- calculateSignalLevel() method, WifiManager, 247–248**
- Calibration, sensor, 229–230**
- Call button, phone calls, 220–221**
- Calls class, 221–222**
- CallVoidMethod() function, exceptions with native code, 383**
- Camera, positioning and animating map, 266–268**
- Camera class**
 - assigning still images as wallpaper, 199
 - capturing still images, 192–196
 - choosing among devices, 199–200
 - common parameters, 197
 - configuring settings, 196
 - face detection, 203–204
 - sharing images, 198
 - working with multimedia, 191–192
 - zooming, 197

- CameraPosition object, 267**
- CameraSurfaceView class, 192–196**
- Campaign tracking, Google Play, 292**
- cancel() method, notifications, 83**
- cancelDiscovery() method, Bluetooth devices, 238**
- canDetectOrientation() method, screens, 136**
- Canvas object**
 - drawing bitmaps, 313
 - drawing on screen with Paint and, 305–309
 - hardware acceleration and, 325
 - understanding, 307
- capture() method, Camera, 196**
- CardView, L Developer Preview, 462**
- CCS (Cloud Connection Server), Google, 272, 274**
- Chaining methods, Java, 501**
- CHANGE_WIFI_STATE permission, 246**
- Character encodings, internationalization, 445**
- CheckJNI tool, 460**
- Chess Utrecht font, 311–312**
- Chrome DevTools, debugging WebView, 187**
- Chromium rendering engine, WebView, 175**
- Circles, drawing, 319–320**
- Classes, Java**
 - documentation, 500
 - method chaining in builder-style, 501
 - working with inner, 503–504
- Classic Bluetooth. See also Bluetooth, 235–236**
- cleanupgl() method, OpenGL ES, 366**
- clear() method, wallpaper, 199**
- Clearing log, 476**
- Click events, long click, 121–122**
- Client, integrating GCM on Android, 273–274**
- Clipboard framework, textual input, 118–119**
- ClipboardManager, 119**
- ClipData object, 119, 134**
- close() method**
 - Cursor management, 41
 - SQLite database, 46
- Cloud Connection Server (CCS), Google, 272, 274**
- Cloud Save, game data, 299**
- Cloud to Device Messaging (C2DM), 271**
- Code**
 - ProGuard, 450–452
 - secure practices, 450
- Color**
 - 3D objects, 355–356, 358–360
 - building simple styles, 107–109
 - Google TV, 157
 - indicator lights for notifications, 84–85
 - L Developer Preview, 462
 - Paint, 307–309
- Columns**
 - building content provider, 56–57
 - data types for SQLite database, 492
 - raw queries, 45
 - SQLite database calculated, 496–497
- compare() method, PhoneNumberUtils, 215**
- compareSignalLevel() method, WifiManager, 247–248**
- Compatibility**
 - development environments, 6
 - notifications and, 78–79
 - OpenGL ES device, 346–347
- Complex queries, 43–44**
- Composite primary keys, SQLite database tables, 493–494**
- Concurrency, SQLite limitations, 490**
- CONFIG_CHECK_GL_ERROR flag, initializing GLS, 351**

Configuration Activity, App Widgets, 395**Connections**

- between Bluetooth devices, 238–239
- buffering of large bitmaps over slow, 170
- fused location provider, 261
- geocoding requiring network, 260
- implementing remote interface, 27
- retrieving Android network status, 171–173
- to Service, 20–21
- to SQLite database, 486–487

ConnectivityManager class, 171–173**ConsoleMessage class, WebKit API, 182****Constants, Sensor class, 227****Content**

- loading into WebView, 176–178
- selling through billing APIs, 278
- web extensions, 183

Content providers

- acting as, 55
- data columns, 56–57
- data URI, 56
- delete() method, 60–61
- enhancing applications, 62–65
- getType() method, 61–62
- insert() method, 59
- interface, 55–56
- query() method, 57–58
- quiz Q & A, 65, 519
- searches in applications, 417–420
- sharing images, 198
- SQLite database acting as, 485
- update() method, 59–60
- updating manifest file, 62
- UriMatcher class, 58–59
- UserDictionary, 118

ContentResolver

- retrieving content provider data, 64–65
- sharing audio, 206
- sharing images, 198

Content type filters, 410–413**ContentValues object, update(), 38–39****Context menu, long-click events, 121****Contextual action mode, action bars, 105–106****Contractible notifications, 88–90****Controls**

- for accessibility, 140
- binding data to, 50–53
- hardware acceleration, 325

Conventions, used in this book, 7**CookieManager class, WebKit API, 182****Copying**

- ADB commands for files, 472
- data to system clipboard, 119

CREATE_AUTHOR_TABLE, SQL, 37**CreateBeamUriCallback interface, Android Beam, 245****createBitmap() method, 313****CreateNdefMessage() method, Android Beam, 242–243****CREATE TABLE SQL statement, 37****CREATE TRIGGER SQL statement, 38****Credentials, user account, 428****Criteria object, device location, 255****CursorLoader, search Activity, 423****Cursor object**

- binding data to controls, 50–53
- content provider query(), 57
- querying SQLite databases, 41–42
- retrieving content provider data, 64

Custom binaries, installing, 481–482

CustomGL2SurfaceView class, OpenGL ES 2.0, 370

Customizing

- action bar, 101–103
- backup services, 431
- Home screen wallpaper, 192, 199
- notifications, 86–88
- software keyboards, 117
- typefaces, 310–312
- View controls for accessibility, 140

Custom Locale application, 442

CustomRenderer class, OpenGL ES 2.0, 370

CycleInterpolator, animation, 341

Cygnin 1.7 or later, Android NDK, 378

D

Dalvik Debug Monitor Service. See DDMS (Dalvik Debug Monitor Service)

Data

- adding to SQLite database, 492
- altering/updating in SQLite database, 495
- backup services, 430–435
- binding to application user interface, 48–53
- building content provider, 56–57
- collecting for Google Analytics, 287
- gathering statistical, 292–293
- Google Analytics tracking
 - e-commerce, 290–292
- reading sensor, 228–229
- retrieving content provider, 64–65
- synchronizing, 429–430

Data adapters, 50–53

dataChanged() method, backups, 434

Data sources, 16

Data types, 492

DateFormat utility class, localizing language, 445

Daydream (screen saver), 408–410

DDMS (Dalvik Debug Monitor Service)

- dumping database contents, 489
- inspecting database files, 485
- simulating call states, 213

deactivate() method, Cursor, 41

Debug certificate, 263–265

Debugging

with ADB. *See* **ADB (Android Debug Bridge)**

- Android NDK applications, 379
- Android Studio applications, 515–517
- ART runtime improvements, 460
- database files with DDMS, 485, 489
- on devices connected to USB hardware, 239
- error reports after ProGuard tool, 452
- WebView control, 187

DecelerateInterpolator, animation, 341

Default Keypad Reference, Android Studio, 517

Default messaging application, 215–216

delete() method, content provider, 60–61

deleteDatabase() method, SQLite, 46

Design

- accessibility. *See* **Accessible applications**
- useful notifications, 91–92
- user interface. *See* **User interface**

Design view, Android Studio, 514

Development environments, used in this book, 5–6

Devices

- attracting new types of users, 153
- copying files to, 472
- determining locale of, 444

- determining location of, 254–256
- directing ADB commands to specific, 470
- disallowing sqlite3, 487
- flexible user interfaces for, 152
- indicator light support, 84–85
- input methods, 115
- installing custom binaries, 481–482
- listing all Android connected, 469
- live wallpaper warning, 406
- notification support, 78
- OpenGL ES compatibility with, 346–347
- presumptions about features, 151–152
- retrieving files from, 472
- sensor support, 227
- testing sensors on physical, 227
- using Multimedia APIs. *See* **Multimedia APIs**
- devices command, adb command, 469**
- Dimensions, flexible user interface, 152**
- dimens.xml file, styles, 107–109**
- disable() method, screen orientation, 136**
- Discovering devices, Bluetooth, 237–238**
- doAlert() function, web extensions, 185**
- doConsoleLog() function, web extensions, 185**
- Documentation, Java, 500**
- doInBackground() method, AsyncTask, 13**
- Domain-specific language (DSL), Gradle, 513**
- doServiceStart() method, 22**
- doSetFormText() function, web extensions, 185–186**
- Dot (.), sqlite3 commands, 486**
- doToast() function, web extensions, 185**
- Double-tap gestures, 124–125, 128–129**
- DragAndDropDemo.java, 134**
- Drag-and-drop framework, 134**
- Dragging, single-touch gesture, 124**
- DragShadowBuilder class, 134**
- draw() method, coloring vertices, 355–356**
- Drawable animations, 329–331**
- Drawable object, 171**
- Drawable resources, L Developer Preview, 462**
- drawFrame() method, 366–367, 372, 384–385**
- Drawing 2D objects**
 - with canvases and paint, 305–308
 - overview of, 305
 - with radial gradients, 309
 - with sweep gradients, 309
 - with text, 310–312
- Drawing 3D objects**
 - coloring vertices, 355–356
 - complex objects, 356–358
 - defined, 345
 - lighting scene, 358–359
 - setting up screen, 353–354
 - texturing objects, 359–362
 - vertices, 353–355
- Drawing on screen**
 - 2D objects. *See* **Drawing 2D objects**
 - 3D objects. *See* **Drawing 3D objects**
- DreamService class, Daydream, 409**
- droid_wallpaper.xml, 407**
- DROP TABLE command, SQLite database, 46, 497**
- DSL (domain-specific language), Gradle, 513**
- dump command, sqlite3, 488**
- dumpsys batterystats, Project Volta, 461**
- duration attribute, property animation, 338**
- duration property, tweened animations, 333**

E

E-commerce overview reports, Google Analytics, 290–292

EditText control

- basic searches, 417
- building styles, 106–109
- choosing software keyboard, 116–117
- listening for focus changes, 122–123
- making phone calls, 220–221
- speech recognition, 141
- working with phone numbers, 215

EGL (Embedded-System Graphics Library), 345, 350–352

EGL10.EGL_DEFAULT_DISPLAY, GLS, 351

eglDestroyContext() method, OpenGL ES, 366

eglDestroySurface() method, OpenGL ES, 366

eglMakeCurrent() method, OpenGL ES, 366

eglTerminate() method, OpenGL ES, 366

elevation attribute, CardView, 462

Embedded-System Graphics Library (EGL), 345, 350–352

Emulators

- allowing sqlite3 command, 487
- copying files to, 472
- installing custom binaries, 481–482
- locating your, 256
- network settings, 173
- nonsupport for simulating hardware sensors, 225
- retrieving files from, 472
- using ADB shell to start/stop, 471–472
- using ADB to list all Android connected, 469

enable() method

- Bluetooth, 237
- screen orientation, 136

endTransaction() method, SQLite databases, 40

Enunciation, speech recognition, 142

Error checking, conventions used in this book, 7

Evaluator classes, property animation, 337

Event handling

- View control, 140
- WebView control, 179–180
- XML Pull Parser, 167

Events

- filtering logs of certain severity, 475
- gathering statistics, 292
- Google Analytics Dashboard reports, 288
- interacting with OpenGL ES and Android, 362–365
- listening for focus changes, 122–123
- listening for long clicks, 121–122
- listening for on entire screen, 120–121
- listening for touch mode changes, 119–120
- live wallpaper responding to user, 406
- logging e-commerce, in Google Analytics, 290–291
- logging in Google Analytics, 287
- stress testing applications with monkey, 478–481

ExceptionDescribe() function, native code, 383

Exception handling

- conventions used in this book, 7
- networking code and, 165

ExceptionOccurred() function, native code, 383

Exceptions, with native code, 384

execSQL() method, SQLite database, 37

execute() method

- executing tasks in parallel, 14
- launching AsyncTask, 13–14
- parallel execute, 14

executeOnExecutor() method, parallel execute, 14**Exerciser Monkey tool, ADB shell, 478–481****Expandable notifications, 88–90****exported attribute, <intent-filter> tag, 74****Exporting, database/data with sqlite3, 488****Extending Android application reach**

- acting as content type handler, 410–411
- with App Widgets. *See* **App Widgets**
- with Daydream, 408–410
- determining Intent actions/MIME types, 411–413
- with live wallpapers, 404–408
- overview of, 391–392
- quiz Q & A, 413–414, 524–525

extensions, building web, 182–187**Extract as Style, styles, 111****EXTRA_LANGUAGE_MODEL, RecognizerIntent, 145****EXTRA_PROMPT, RecognizerIntent, 145**

F

Face class, Camera, 203–204**Face detection, 203–204****Features**

- adding to WebView, 176–178
- antipiracy tips, 453
- avoiding presumptions about device, 151–154
- specifying multimedia, 191–192

Field names, tracking database, 47**FileBackupHelper class, 432–434****File Explorer, 485, 489****FileLock, backup helper, 434****File pointers, Cursor objects as, 41–42****Files**

- Android Studio project structure, 512
- executing SQL scripts with sqlite3, 489
- implementing backup helper for, 432
- redirecting log output to, 476
- sending output with sqlite3, 488
- SQLite databases as private, 485
- standard format for database, 485

fillAfter property, moving transformations, 336**Filtering log information, LogCat, 474–475****findViewById() method, web extensions, 183****Finger gestures. *See* Gestures****Fingerprint, map API key, 263–264****Flash**

- optimizing for Google TV, 157
- working with, 187–188

Flickr image, displaying, 170–171**Fling gestures, 124–125, 128****FloatBuffer() method, drawing vertices, 354–355****FloatEvaluator class, property animation, 337****Folders, Android Studio projects, 512****Fonts**

- conventions used in this book, 7
- customizing typefaces, 310–312
- enabling accessibility, 140
- modifying WebView control, 178
- using default, 310

for-each loops, looping infinitely, 502**Foreground, designing notifications, 91****Foreign keys, SQLite database, 37–38, 493–494****for loops, looping infinitely, 501–502**

- format() method, locale strings, 445
- formatJapaneseNumber() method, phone numbers, 215
- formatNumber() method, phone numbers, 215–216
- Fragment
 - asynchronously loading data to, 16
 - gathering statistics, 292
 - launching Google Maps, 265–266
 - for tablets, 155
 - for user interfaces, 152
- Frame-by-frame animations, 329–331
- Frame rate, OpenGL/application threads, 363–364
- FrameLayout, 349
- Freemium business model, 277–279
- fromAlpha value, alpha transparency transformations, 334–335
- fromDegrees property, transformations, 335
- fromXDelta, fromYDelta values, transformations, 336
- fromXScale, fromYScale values, transformations, 335–336
- Front-facing camera, 199–200
- FTS3 extension, SQLite, 420
- Functions, Service for routine/regular, 19
- Fused location provider, Google location services, 260–261

G

- GameHelper class, Google Play game services, 296–297
- Gaming
 - design challenges of, 154
 - Google Play. *See* **Google Play game services**
 - secure coding practices, 450
- Garbage collection (GC), ART runtime, 460

- GC (garbage collection), ART runtime, 460
- GCM (Google Cloud Messaging)
 - alternatives to, 274–275
 - incorporating into applications, 273–274
 - limitations of, 272–273
 - message flow, 272
 - overview of, 271–272
 - quiz Q & A, 275, 522
 - sample applications, 274
 - signing up for, 273
- Geocaching, 260
- Geocoding, 256–260
- Geofencing APIs, 262
- GeomagneticField class, true north, 230
- gesture package, 124
- GestureDetector class, 123–129
- GestureListener class, 129, 132
- GestureOverlayView, 124
- Gestures
 - common multitouch, 133
 - common single-touch, 124–129
 - detecting user motions within View, 123–124
 - natural-looking, 133
 - user input with, 123
 - using drag-and-drop framework, 134
- getAccessoryList() method, USB, 240
- getAccountByType() method, user accounts, 428
- getActualDefaultRingtoneUri() method, 209
- getAddressLine() method, geocoding, 258
- getAllocationByteCount() method, bitmaps, 314
- getAllProviders() method, device location, 254
- getApiClient() method, Google Play game services, 297

- getAuthToken() method, user accounts, 428
- getAvailableLocales() method, Locale class, 444
- getBestProvider() method, device location, 254–255
- getBondedDevices() method, Bluetooth, 237
- getCallState() method, 212–213
- getClipData() method, drag-and-drop, 134
- getConfiguredNetworks() method, WifiManager, 248
- getDefault() method
 - Locale class, 444
 - SmsManager, 218
- getDefaultAdapter() method, Bluetooth, 236–237
- getDefaultSensor(), 227
- getDesiredMinimumHeight() method, wallpaper, 199
- getDesiredMinimumWidth() method, wallpaper, 199
- getDeviceList() method, USB host, 241
- getDrawable() method, wallpaper, 199
- getFragmentManager() method, Google Maps, 265–266
- getFromLocation() method, geocoding, 257
- getFromLocationName() method, geocoding, 259
- getHeight() method, Canvas, 307
- getHolder() method, SurfaceView, 348
- getLocality() method, geocoding, 257
- getMap() method, Google Maps, 265–266
- getMaxAddressLineIndex() method, geocoding, 258
- GetMethodID() function, native code exceptions, 383
- getNumberOfCameras() method, 200
- getOrientation() method, 230
- getRoaming() method, 214
- getScaleFactor() method, multitouch gestures, 132
- getScanResults() method, Wi-Fi state, 247
- getSettings() method, WebView, 178
- getString() method, Cursor, 42
- getSupportFragmentManager() method, Google Maps, 265–266
- getSystemService() method
 - copying/pasting data, 119
 - determining device location, 254
 - NotificationManager, 79
 - retrieving Android network status, 172
 - USB accessories, 240
 - working as USB host, 241
- getTextBounds() method, measuring screen, 312
- getType() method, content providers, 61–62
- getWidth() method, Canvas, 307
- GitHub account, PayPal billing APIs, 280
- GL (Graphics Library), 345
- GL_COLOR_ARRAY, vertices, 355–356
- GL_COLOR_MATERIAL, lighting scenes, 358–360
- glColorPointer() method, vertices, 355–356
- glCompileShader() method, OpenGL ES 2.0, 372
- GLDebugHelper class, GLS, 351–352
- glDrawArrays() method, vertices, 353–355
- glDrawElements() method
 - complex 3D objects, 356–357
 - vertices, 353–355
- GL_LINE_LOOP, drawing 3D objects, 357
- Global Positioning System (GPS), 21–25, 254
- Global searches, enabling, 424–425
- glRotatef() method, drawing 3D objects, 353
- glShaderSource() method, OpenGL ES 2.0, 372

GLSurfaceView, 347, 366–369

GL_TEXTURE_COORD_ARRAY state, 3D objects, 361–362

gluLookAt() method, drawing 3D objects, 353

gluPerspective() method, OpenGL ES, 352

glUseProgram() method, OpenGL ES 2.0, 372

GLUT (OpenGL Utility Toolkit), 352

GLUtils.texImage2D() method, 3D objects, 361–362

GL_VERTEX_ARRAY state, drawing vertices, 355

glVertexPointer() method, drawing vertices, 353–355

Gmail, Google Cloud Messaging service, 272

gms.common.api package, Google Play game services, 296–297

GNU Awk, Android NDK, 378

GNU Make 3.81, Android NDK, 378

go() method, scene state transitions, 342

Google account, creating, 272, 283–285

Google Analytics

adding library to Android IDE project, 286–287

Admin section, 284–285

collecting data, 287

creating Google account, 283–285

gathering e-commerce data, 290–292

gathering statistics, 292–293

logging different events, 287

overview of, 283

protecting user privacy, 293

quiz Q & A, 293–294, 523

Reporting section, 284–285

tracking ad/market referrals, 292

Google Analytics Dashboard, 288–290

GoogleApiClient object, 297

Google APIs for Android

in-app billing. *See* **In-app billing APIs**

enabling application statistics. *See* **Google Analytics**

GCM. *See* **GCM (Google Cloud Messaging)**

location and maps. *See* **Location and map APIs**

Google Cast SDK, 209

Google Cloud Messaging. *See* **GCM (Google Cloud Messaging)**

Google Developer Console, 263, 273

Google location services APIs, 260–262

Google Maps Android API v2

creating long-lasting/shareable debug key, 264–265

launching with location URI, 263–265

mapping fragments, 265–266

marking spot, 265–266

overview of, 262

positioning/animating map camera, 266–268

Google Play

adding services library to Android IDE project, 286–287

Android location APIs on devices without, 253

in-app billing APIs, 279

applications for tablets, 155

campaign tracking, 292

GCM for Android, 272

Google location services APIs, 260–262

publishing applications for foreign users, 446

publishing native Google TV apps, 157

translation services, 445

Google Play Developer Console, 295–296, 446

Google Play game services

- achievements, 297–298
- antipiracy, 299–300
- getting started, 295–296
- incorporating into applications, 296–297
- leaderboards, 298
- multiplayer gaming, 299
- overview of, 295
- quiz Q & A, 300, 523
- quota and rate limiting, 298
- saving game data, 299

google-play-services_lib project, 286**Google TV, 155–158****Google Wallet, 446****GPS (Global Positioning System), 21–25, 254****GpsSatellite class, 260****GpsStatus class, 260****GpsStatus.Listener class, 260****GPXService class, 20–21, 26–28****Gradients, Paint, 307–309****Gradle build system, Android Studio, 512–513****Graphical Layout editor, styles, 111****Graphics**

- alternative resources for, 153
- Android NDK. *See* **Android NDK (Native Development Kit)**
- animation. *See* **Animation**
- designing layouts with WebView, 176
- designing UI with stretchable, 152
- L Developer Preview optimization, 460–461
- using space on big landscape screens, 153–154

Graphics, 2D applications

- drawing on screen, 305–309

- hardware acceleration features, 324–326
- overview of, 305
- quiz Q & A, 326, 523
- working with bitmaps, 312–315
- working with shapes, 315–324
- working with text, 310–312

Graphics, 3D applications

- cleaning up OpenGL ES, 365–366
- coloring vertices, 355–356
- drawing more complex objects, 356–358
- drawing vertices, 353–355
- interacting with Android views/ events, 362–365
- lighting scene, 358–360
- live wallpapers, 404–408
- OpenGL ES 2.0, 369–373
- OpenGL ES 3.0, 373–374
- OpenGL ES, APIs in Android SDK, 347
- OpenGL ES, handling tasks, 347–353
- OpenGL ES, working manually, 345–347
- quiz Q & A, 374, 524
- texturing objects, 359–362
- using GLSurfaceView, 366–369

Graphics Library (GL), 345**GridView control, contextual action, 105**

H

Haptic feedback, accessibility, 140**Hardware**

- 2D acceleration features, 324–326
- checking for Bluetooth, 236–237
- limitations of TV application, 465
- optional multimedia device, 191

hardwareAccelerated attribute, 325**Hardware APIs**

- Android Beam, 241–245
- Bluetooth. *See* **Bluetooth**
- quiz Q & A, 248–249, 522
- USB, 239–241
- Wi-Fi, 245–248

Hardware sensors

- acquiring reference to, 227
- batching calls, 230
- calibrating, 229–230
- configuring Android manifest file for, 227
- determining device orientation, 230
- finding true north, 230
- interacting with device hardware, guidelines, 225–226
- monitoring battery, 231–233
- quiz Q & A, 233, 522
- reading data, 228–229
- using device, 226
- working with different, 226–227

Heads-up notifications, L Developer Preview, 463**hello-jni native library, Android NDK, 379****HelloStudio module, Android Studio, 510–512****help command, ACB, 470****Helper methods, WifiManager, 247–248****hint attribute, basic search, 417****historian.par (Battery Historian), Project Volta, 461****Home screen**

- accessing App Widget on, 393
- changing language settings, 442
- Google Analytics Dashboard reports, 288

- installing App Widget to, 394, 400–401
- installing live wallpaper on, 407–408

Host

- App Widget, 392, 401
- working as USB, 241

Host card emulation applications, Android Beam, 245**HTTP**

- GCM for Android, 272, 274
- transferring data to and from network, 164–166

HttpURLConnection, 165–166**I****Icons**

- creating simple text notification, 79–80
- displaying on notification queue, 80–82
- expandable notifications, 89–90
- handling application icon clicks on action bar, 103–104
- as notification component, 79
- updating notifications, 83

IDE (integrated development environment). *See* Android Studio**Identifiers**

- deleting SQLite database records, 39–40
- Notification, 80–81

Images

- camera. *See* **Still images**
- displaying from network resource, 170–171
- storing in database, 53

ImageSwitcher, 171**ImageView background, animation, 330–331, 334**

IMEs (Input Method Editors), 115, 117

Import statement, conventions used in this book, 7

Importing, database/data with sqlite3, 489

In-app billing APIs

Amazon Appstore for Android, 280

antipiracy tips, 453

Google Play, 279

other, 280

PayPal, 280

quiz Q & A, 281, 523

understanding, 277–278

using, 278–279

includeInGlobalSearch attribute, global searches, 425

Index arrays, drawing 3D objects, 356–358

Indicator lights, notifications, 84–85

Indices of table, listing with sqlite3, 487

Infinite loops, Java, 501–502

inflate() method, themes, 112

Inheritance, style, 109–111

Initialization

GLS, 350–352

OpenGL ES, 352

initialLayout attribute, App Widgets, 394

Inner classes, Java, 503–504

Input

Android applications leveraging speech, 139

methods. *See* **User input methods**

Input Method Editors (IMEs), 115, 117

InputStream

connections between Bluetooth devices, 238

displaying images from network resource, 171

reading data from Web, 165

inputType attribute values, software keyboard, 116–117

insert() method

adding data to content provider, 59

records in SQLite database, 38

insertImage() method, sharing images, 198

insertOrThrow() method, SQLite database records, 38

Installation

Android NDK, 378

Android Studio, 507

App Widget to Home screen, 394, 400–401

of custom binaries, 481–482

live wallpaper, 407–408

Lock screen App Widget, 403

using ADB, 473

install command, ADB, 473

Integrated Raster Imaging System Graphics Library (IRIS GL), 345

IntelliJ IDEA, Android Studio based on, 507

Intent action namespace, 67

Intent object

application acting as content filter, 410–413

clearing notifications, 82–83

controlling Service, 25

global searches, 425

implementing remote interface, 26–28

launching browser, 176

launching Google Maps, 263

making phone calls, 220–221

as notification component, 79

receiving phone calls, 221–222

recording speech, 142–145

searching for multimedia, 207–208

textual input, 119

Intent objects, broadcasts

- overview of, 67
- quiz Q & A, 74
- receiving broadcasts, 69–73
- securing application broadcasts, 73–74
- sending broadcasts, 67–68

IntentService class, 30–33**Interacting with Android views/events, 3D graphics, 362–365****Interface. See User interface****Internationalizing applications**

- alternative resources for, 439–442
- language settings, 442–444
- locale support programmatically, 444–445
- localizing language, 439
- publishing applications for foreign users, 446
- quiz Q & A, 446–447, 525
- right-to-left language localization, 445
- translation services via Google Play, 445

INTERNET permission, playing video, 202**Internet, transferring data, 164–166****Interoperability, Android application, 391****Interpolators, 341****IntEvaluator class, property animation, 337****Introduction to this book**

- changes in this edition, 4–5
- contacting authors, 8
- conventions used, 7
- development environments used, 5–6
- questions answered, 3–4
- structure used, 1–3
- supplementary materials, 6
- where to find more information, 6–7
- who should read it, 1

invalidate() method, gestures, 127, 131**IN_VEHICLE, activity recognition APIs, 261****IRemoteInterface, 26–28****IRIS GL (Integrated Raster Imaging System Graphics Library), 345****isAfterLast() method, Cursor, 41–42****isDiscovering() method, Bluetooth devices, 238****isEmergencyNumber() method, PhoneNumberUtils, 215****isHardwareAccelerated() method, View control, 326****IS_RINGTONE flag, 207****isVideoSnapshotSupported() method, 201****Iterating query results, with Cursor, 41–42**

J
Java

- calling native code from, 377–378, 380–381
- supporting Open GL ES 2.0 with Android, 369–373
- susceptible to reverse engineering, 450

Java, for Android developers

- familiarity with Java documentation, 500
- learning Java development tools, 499–500
- learning Java programming language, 499
- quiz Q & A, 505, 526
- understanding Java shorthand, 500–504

Javadocs, 500**JavaScript, 178, 183–187****JavaScriptExtensions class, 184****@JavaScriptInterface annotation, warning, 184**

`javaThrowsException()` method, native code, 383

JIT (just-in-time) compilation, ART runtime, 460

JNI bindings, debugging, 460

JNIEnv object, native code, 382–383

`/jni` subdirectory, NDK project, 379–380

JobScheduler API, Project Volta, 461

Jobs, Steve, 98

Joins, SQLite, 491, 495

`jse.javaMethod()`, web extensions, 184

K

Keyboards. *See* Software keyboards

Keyboard shortcuts, Android Studio, 517

Key handling, GLSurfaceView, 368

Keymap quick reference card, Android Studio, 517

keytool command-line tool, debug key, 264–265

Killing services, 20

kill-server command, ADB server, 470

KitKat (Android 4.4), 6, 508

L

label attribute, basic searches, 417

Language & input settings, 442

Languages

- international. *See* **Internationalizing applications**
- speech recognition services, 146

Latency, controlling network, 173

Latitude

- geocoding locations, 256, 258–260
- geofencing APIs, 262
- launching Google Maps, 263

Launching Android Studio, 508

launchRecognizer value, voice search, 420

launchWebSearch value, voice search, 420

Layout

- applying styles, 106–109
- creating App Widget, 394
- designing devices with flexible, 152
- designing with WebView control, 176
- internationalization via alternative resources, 440–442
- right-to-left language localization, 445
- style inheritance, 109–111
- themes, 111–113
- using RemoteViews, 397
- using space on big landscape screens, 153–154
- web extensions, 183

Layout Editor, Android Studio, 513–515

LBS (location-based services), 145

L Developer Preview, or Android L

- Android TV, 464–465
- improving performance, 460–461
- improving user experience, 461–464
- new APIs added to, 459–460
- quiz Q & A, 465, 526
- setting up SDK, 459

Leaderboards, Google Play game services, 298

LEDs, notifications, 84–85

Library services, Google Analytics SDK for Android, 286–287

`/libs` folders, Android Studio, 512

License Verification Library (LVL), 452–453

Lifecycle

- Activity, 12–14
- broadcast receiver, 69
- Cursor objects, 41
- Service, 20
- Service vs. Activity, 21

Lighting scenes, OpenGL ES, 358–359

Linear gradients, 308

LinearInterpolator, animation, 341

lint tool, accessibility issues, 141

listen() method, TelephonyManager, 213, 214–215

Listeners, using inner classes, 504

Listening

for events on entire screen, 120–121

for long clicks, 121–122

for touch mode changes, 119–120

LISTEN_SERVICE_STATE flag, 213

ListView control

binding data to controls, 50–53

contextual action mode, 105

retrieving content provider data, 64

Live wallpapers

configuring, 406–407

creating, 404–406

installing, 407–408

overview of, 404

loadAndCompileShader() method, OpenGL ES 2.0, 371–372

loadData() method, WebView, 177

Loader class, 12, 16

LoaderManager, 16

loadInBackground() method, search Activity, 423

Loading animations, 334

Loading content, 176–178, 183

Loading images, 314

loadUrl() method, WebChromeClient, 179

LocalBroadcastManager class, 74

Locale

changing language settings, 442–444

implementing programmatically, 444–445

internationalization via alternative resources, 439–442

Localizing application language

changing language settings, 442–444

implementing programmatically, 444–445

overview of, 439

right-to-left, 445

using alternative resources, 439–442

Location and map APIs

Android location APIs. *See* **Android location APIs**

Google location services APIs, 260–262

Google Maps Android API v2, 262–268

GPS, 254–256

quiz Q & A, 268–269, 522

Location-based services (LBS), 145

Location class, Parcelable interface, 26

LocationListener object, 254–255

LocationManager, 254–255, 260

LOCATION_SERVICE, 254

Location URI, launching Google Maps with, 263

Lock, for file backups, 433

Lock screen

App Widgets, 393, 401–403

L Developer Preview improvements, 463

logcat command, 474

LogCat utility, 473–474

Logging

e-commerce events in Google Analytics, 290–291

events in Google Analytics, 287

with LogCat utility, 474–476

Long-click event, listening for, 121–122

Long-form dictation, SpeechRecognizer, 141

Longitude

geocoding locations, 256, 258–260

geofencing APIs, 262

launching Google Maps with location URI, 263

Looping infinitely, Java, 501–502

LVL (License Verification Library), 452–453

M

makeCustomAnimation() method, 341

makeScaleUpAnimation() method, 341

makeThumbnailScaleUpAnimation() method, 341

Map API key, 263–265

Map APIs. See Google Maps Android API v2

MapFragment, 265–266

mapping.txt, 452

Marker, Google Maps, 266–267

Material design, L Developer Preview, 461–463

Material theme, L Developer Preview, 462

Matrix class, transforming bitmaps, 313–314

Measuring text screen requirements, 312

MediaController, 202–203

MediaDrm class, 202

MediaPlayer object, 202–203, 205–206

MediaRecorder object, 200–201, 204–205

MediaRouteProvider APIs, 209

Media router APIs, 209

Media router library, multimedia APIs, 209

MediaScannerConnection class, 198

MEDIA_SEARCH, 207–208

MediaStore content provider, 198, 206

Memory, bitmap optimization, 313–314

MenuInflater, contextual action mode, 104

Message flow, Google Cloud Messaging, 272, 273–274

Metadata, NotificationListenerService, 91

<meta-data> manifest tag

Android Backup Service, 431

App Widgets, 399

Google Cloud Save, 299

Google Maps Android, 265

search, 423–424

Methods

building content provider, 57–62

chaining in Java, 501

MIME types

application acting as content filter, 410–413

building content provider, 61–62

overview of, 411

minSdkVersion, action bars, 101

MMS (Multimedia Messaging Service), 216–217

monkey application, 472, 478–481

monospace font

conventions used in this book, 7

drawing on screen, 310

MotionEvent object, gesture detectors, 123

Mouse-overs, WebView control, 178

moveToFirst() method, Cursor, 41–42

moveToNext() method, Cursor, 41–42

Moving transformations, 336

mSaveText, listening for focus changes, 122–123

MultiChoiceModeListener(), contextual action mode, 105

Multimedia APIs

media router library, 209

overview of, 191

playing audio, 205–206

playing video, 202–203

Multimedia APIs (*continued*)

- quiz Q & A, 209–210, 521
- recording audio, 204–205
- recording video, 200–201
- ringtones, 208–209
- searching for multimedia, 207–208
- sharing audio, 206–207
- working with, 191–192
- working with face detection, 203–204

Multimedia APIs, still images

- assigning as wallpaper, 199
- capturing with camera, 192–196
- choosing among device cameras, 199–200
- common camera parameters, 197
- configuring camera mode settings, 196
- sharing images, 198
- zooming camera, 197

Multimedia Messaging Service (MMS), 216–217**Multiplayer gaming, Google Play game services, 299****Multiple processors, 16****Multiple users, creating on devices, 428–429****Multitouch gestures, 123, 129–133****MyOnClickListener class, using inner classes, 504**

N

name attribute, Google Cloud Save, 299**Naming conventions**

- applications for tablets, 155
- content provider URI, 56

National Geospatial-Intelligence Agency (NGA), World Magnetic Model, 230**Native Android applications, Adobe Air and, 188****NativeBasicsActivity.java, 380–381****native_basics.c file, 381****Native code. See Android NDK (Native Development Kit)****Native Development Kit. See Android NDK (Native Development Kit)****Native Google TV apps, 157–158****Native libraries, Android NDK**

- building own NDK project, 380
- building sample application, 379
- improving performance, 384–385
- only on Android 1.5 and higher, 377–378

Navigation

- detecting gestures, 123
- optimizing web applications for Google TV, 157
- user interface. *See* **Action bars**

NdefMessage, 242–243**NDEF (NFC Data Exchange Format) Push over NFC, 241–243****NDK. See Android NDK (Native Development Kit)****ndk-build script, building own NDK project, 380****Nesting transactions, SQLite, 40, 491****NetworkInfo objects, Android network status, 172****Networking APIs**

- accessing Internet, 164–166
- Android network status, 171–173
- asynchronous operations, 12, 167–171
- mobile networking, 163
- overview of, 163
- parsing XML from network, 166–167
- quiz Q & A, 173, 521
- StrictMode, 164

NetworkOnMainThreadException, 164

Network operator name, requesting service information, 214

Network resources, displaying images, 170–171

New Project creation wizard, Android Studio, 509–512

NfcAdapter class, Android Beam, 242–243

NFC Data Exchange Format (NDEF) Push over NFC, 241–243

NGA (National Geospatial-Intelligence Agency), World Magnetic Model, 230

Noise, notifications, 86

Nonprimitive types, 53, 170–171

Nonprofit Khronos Group, OpenGL ES, 345

Normal broadcasts, 67

Notification.Builder() class, 78, 80–81, 83

NotificationCompat library, 78

NotificationCompat.Builder class, 79, 85–86, 88–90

NotificationListenerService, 91

NotificationManager object, 79, 80–81, 83

Notification queue

- clearing, 82–83
- customizing, 86–88
- expandable and contractible notifications, 88–90
- setting priority, 90–91
- updating, 81–82
- working with, 80–81

Notifications

- blinking lights, 84–85
- clearing, 82–83
- communicating data to user, 24
- compatibility, 78
- components of simple, 79
- customizing, 86–88
- designing useful, 91–92
- expandable and contractible, 88–90

- GCM for Android, 271–272
- L Developer Preview enhanced, 463–464
- making noise, 86
- notification listener, 91
- notification queue, 80–81
- notifying user, 77–78
- quiz Q & A, 92, 520
- setting priority, 90–91
- simple text with icon, 79–80
- sounds of system events, 208
- with status bar, 78–79
- updating, 81–82
- using NotificationManager service, 79
- vibrating phone, 84
- for wearables, 158

notify() method, 79–81

notifyBuilder variable, 79

O

ObjectAnimator class, property animation, 337

Offload processing, 12, 20

onActionItemClicked() method, contextual action mode, 104

onActivityResult() method, recording speech, 145

onAnimateMove() method, fling gestures, 128

onAnimateStep() method, fling gestures, 128

onBackup() method, 431, 433–434

ON_BICYCLE, activity recognition APIs, 261

onBind() method

- creating Service, 21
- implementing remote interface, 26–28

onClick() method

- building basic action bars, 100
- building web extensions, 185

`onClick()` method (*continued*)

- geocoding locations, 258
- playing audio, 205
- recording audio, 204–205
- recording video, 201

`onConnected()` method, fused location provider, 261**`onConnectionFailed()` method, fused location provider, 261****`OnConnectionFailedListener`, 261****`onConsoleMessage()` method, web extensions, 183****`OnCreate()` method**

- action bars, 104–105
- `AsyncTask` class, 13
- data for Google Analytics, 287
- Google Play game services, 297
- remote interface, 27
- search Activity, 423
- Service, 20–21, 25
- `SQLiteOpenHelper`, 47–48
- `Thread` class, 15–16
- wallpaper Service, 405
- web extensions, 183–184

`onCreateLoader()` method, 16**`onCreateOptionsMenu()`, search, 420–421****`onDelete()` method, `AppWidgetProvider`, 396****`OnDestroy()` method**

- Cursor management, 41
- data for Google Analytics, 287
- Service, 20–21, 24–25
- wallpaper Service, 405

`onDisabled()` method, `AppWidgetProvider`, 395**`onDisconnected()` method, fused location provider, 261****`onDoubleTap()` method, gestures, 124, 128–129****`onDoubleTapEvent()` method, gestures, 124–125****`onDown()` method, gestures, 124, 128–129****`onDowngrade()` method, `SQLiteOpenHelper`, 47–48****`onDraw()` method**

- `Canvas` object, 305–306
- single-touch gestures, 126

`onDrawFrame()` method

- Android NDK, 385
- `GLSurfaceView`, 367–368
- OpenGL ES 2.0, 372

`onEnabled()` method, `AppWidgetProvider`, 395**`onFaceDetection()` callback event, 203–204****`onFling()` method, gestures, 124, 129****`OnFocusChangeListener` class, 122–123****`ON_FOOT`, activity recognition APIs, 261****`OnGlobal LayoutListener` class, 121****`onInit()` method, TTS, 146****`OnInitListener` interface, TTS, 146****`onJsBeforeUnload()` method, `WebChromeClient`, 179****`onKeyDown()` method, OpenGL ES, 364–365, 368****`onKeyUp()` method, OpenGL ES, 365****`onLoaderReset()` method, `Loaders`, 16****`onLoadFinished()` method, `Loaders`, 16****`onLocationChanged()` method, device location, 255–256****`onLocationChanged()` method, geocoding location, 257****`onLongPress()` method, `GestureDetector`, 124****`onMove()` method, gestures, 128–129****`onNewIntent()` method, search Activity, 423****`onOpen()` method, `SQLiteOpenHelper`, 47–48****`onOptionsItemSelected()` method, action bar, 104**

- onOrientationChanged() method, screen, 136
- onPause() method, Cursor, 41
- onPause() method, WebView, 179–180
- onPerformSync() method, sync adapters, 429
- onPostExecute() method, AsyncTask, 13
- OnPreDrawListener class, 120
- onPreExecute() method, AsyncTask, 13
- onProgressUpdate() method, AsyncTask, 13
- onReceive() callback method, broadcasts, 69, 71
- onResetLocation() method, gestures, 128–129
- onRestore() method, 431, 433–434
- onResume() method
 - avoiding logging of, 292
 - Cursor management, 41
 - WebView state, 179
- onScaleBegin() method, multitouch gestures, 132
- onScale() helper method, multitouch gestures, 131
- onScroll() method, gestures, 124, 129
- onSensorChanged() method, 228–229
- onServiceConnected() method, remote interface, 27
- onServiceDisconnected() method, remote interface, 27–28
- onShowPress() method, gestures, 124
- onSingleTapConfirmed() method, gestures, 124
- onSingleTapUp() method, gestures, 124
- onStartCommand() method, Service, 20–22, 25
- onStart() method, Service, 20–22, 25
- onSurfaceChanged() method, wallpaper Service, 405
- onSurfaceCreated() method, wallpaper Service, 405
- onSurfaceCreate() method, OpenGL ES 2.0, 370
- onSurfaceDestroyed() method, wallpaper Service, 405
- onTouchEvent() method, gestures, 123, 125–127, 131
- onTouchEvent() method, wallpaper Service, 405
- onTouchModeChanged() method, 120
- OnTouchModeChangeListener class, 120
- onUpdate() method, App Widget, 395–396, 398
- onUpgrade() method, SQLiteOpenHelper, 47–48
- onVisibilityChanged() method, wallpaper, 405
- OpenGL ES
 - cleaning up, 365–366
 - ensuring device compatibility, 346–347
 - GLSurfaceView, 366–369
 - handling tasks manually, 347–353
 - initializing, 352–353
 - leveraging in Android, 346
 - OpenGL ES 2.0, 369–373
 - OpenGL ES 3.0, 373–374
 - overview of, 345
 - using APIs in Android SDK, 347
- OpenGL ES 3.1, 460–461
- OpenGL rendering pipeline, 2D graphics, 321
- OpenGL Utility Toolkit (GLUT), 352
- openOrCreateDatabase() method, SQLite database, 36
- Operations, asynchronous networking, 167–171
- Options menu resource file, action bars, 100–101
- Ordered broadcasts, 67

Orientation

- alternative resources for, 153
- changing screen, 134–136
- determining device, 230
- developing tablet applications, 155
- using space on big landscape screens, 153–154

OrientationEventListener class, 135–136**OutOfMemoryError, bitmap optimization, 313–314****Output, speech, 139****OutputStream, Bluetooth devices, 238****Ovals, drawing, 319–320****OvalShape object, 319–320****Overflow menu icon, action bars, 101****OvershootInterpolator, animation, 341**

P

Paint object

- anti-aliasing, 307
- drawing text, 309
- gradients, 307
- hardware acceleration and, 325
- setting properties via XML, 316
- styles, 307
- understanding, 307
- working with canvases and, 305–306
- working with color, 307

Parallel execute, 14, 16**Parameters class, camera, 196–197****Parameters, NDK project, 381–382****Parcelable class, 26, 28–30****Parents, style inheritance, 109–111****Parsing XML from network, 166–167****Password protection, software piracy, 450****Pasting data, from system clipboard, 119****Paths, drawing, 322–324****PathShape, 323–324****pause Timers() method, WebView, 180****PayPal billing APIs, 280****peek Drawable() method, wallpaper, 199****PendingIntent, 88–90, 260****Percentage, of battery use, 233****Performance optimization**

- 2D bitmaps, 313
- 3D graphics in Android NDK, 384–385
- L Developer Preview, 460–461

Permissions

- Bluetooth, 235
- camera, 192
- content provider access, 64
- fused location provider, 261
- location of device, 254–255
- making phone calls, 220
- monitoring battery use, 231
- monitoring Wi-Fi state, 246
- phone state information, 212
- recording video, 201, 205
- sending broadcasts, 67–68
- sending/receiving SMS messages, 218
- setting wallpaper, 199
- sounds of system events, 209
- using Google Analytics SDK for Android, 286
- using GPS in your applications, 254
- vibration with notifications, 84
- video from Internet resource, 202
- WebView control, 175–176
- Wi-Fi Direct on Android, 246
- working with SIP, 221

permitAll() method, skipping StrictMode, 17, 164**Persistent databases, 46–48**

Personalizing devices, 199, 208–209
Phone calls, 220–222
PhoneNumberUtils class, 215–216, 445
PhoneStateListener, 214–215
Pinch-to-zoom, multitouch gestures, 129–130
Plug-ins, WebView, 178
postDelayed() method, View class, 15
post() method
 OpenGL ES, 362–364
 View class, 15–16
Power issues
 battery life, 231–233, 461
 Daydream, 408
 live wallpaper, 406
Precision updates, AppWidgetProvider, 396
Preferences, backing up shared, 432
PrefListenerService class, 398
Press-and-hold action, long-click events, 121
Preview All Screen Sizes, Android Studio, 514
Preview controls, Android Studio, 515
previewImage field, App Widgets, 394
Preview window, Android Studio, 514–515
Pricing, antipiracy tips, 454
Printing debug information, 478
Priorities, setting notification, 90–91
Privacy, 74, 408
Processors, 16
Programmatically
 defining property animation, 339–341
 defining shape drawables, 316–317
 defining tweened animations, 333
 implementing locale support, 444–445
ProgressBar control, 13, 202–203, 220
ProGuard, 287, 450–452
proguard-project.txt file, 451
Projects, Android Studio, 509, 512

Project Volta, L Developer Preview, 461
Properties
 SQLite database, 37
 typeface, 310
Property animation
 defined, 329
 defining as XML resources, 337–339
 defining/modifying programmatically, 339–341
 working with, 336–337
propertyName attribute, property animation, 338
Provider(s)
 account, 428
 App Widget, 392–396
 content. *See* **Content providers**
 fused location, 260–261
 location, 254–256
 SMS, 217
 Telephony API service, 214
Publishing applications for foreign users, 446
publishProgress() method, AsyncTask, 13
pull command, retrieving files, 472
Pull Parser, XML, 166–167
push command, copying files, 472
Push messaging services, 271–273, 275

Q

Queries, paired Bluetooth devices, 237
Queries, SQLite database
 complex queries, 44
 to multiple tables, 495
 overview of, 40–41
 raw queries, 45
 with SELECT, 493
 simple queries, 43–44

Queries, SQLite database *(continued)*

- subqueries for calculated columns, 497
- using calculated columns, 496–497
- using sqlite3 to test, 490
- working with cursors, 41–42

query() method

- content provider, 57–58
- enabling search suggestions, 419
- executing simple queries, 43–44
- retrieving content provider data, 64

Questions, answered in this book, 3–4**queueEvent() method, GLSurfaceView, 368****Quick Search Box, global searches, 424–425****Quick-Start Guides**

- ADB. *See* **ADB (Android Debug Bridge)**
- Android Studio. *See* **Android Studio**
- SQLite. *See* **SQLite databases**

Quotas, Google Play game services, 298

R

Radial gradients, 309**Rate limiting management, Google Play game services, 298****Raw HTML, WebView control, 177****rawquery() method, 45****readFromParcel() method, Parcelable class, 29****Reading**

- data from Web, 164–165
- sensor data, 228–229
- text to user, 145–147

READ_PHONE_STATE permission, 212**readText() method, text-to-speech, 147****Real-time multiplayer APIs, Google Play game services, 299****<receiver> tag, App Widgets, 399****Receiving**

- Android Beam messages, 243–244
- broadcasts, 69–73
- phone calls, 221–222
- SMS messages, 218

RecognizerIntent intent, recording speech, 142–145**Reconfigure() method, bitmaps, 314****Recording**

- audio, 204–205
- video, 200–201

Records

- deleting SQLite database, 39–40
- inserting SQLite database, 38
- query results corresponding to returned, 41
- updating SQLite database, 38–39

recordSpeech() method, speech recognition, 145**Rectangles, drawing, 318–319****RectEvaluator class, property animation, 337****RectShape object, 318****recycle() method, transforming bitmaps, 313****RecyclerView, L Developer Preview, 462****Referential integrity, SQLite limitations, 491****registerListener() method, sensors, 228–230****registerReceiver() method, sticky broadcasts, 67****Registration**

- of applications using ADB, 431
- of backup agent in manifest file, 434
- receiving broadcasts and, 69–71

Reinstallation, of applications using ADB, 473**release() method, playing audio, 206****Remote backup service, 430–431****Remote interface, 19, 26–28**

RemoteViews class
 App Widgets, 396–400
 customizing notifications, 86–88

remove() method, SQLite database records, 39–40

Renderer class, GLSurfaceView, 366–369

RenderScript
 Android NDK vs., 385–386
 computing with, 385
 deprecated in Android 4.1, 374

repeatCount attribute, property animation, 338–339

repeatMode attribute, property animation, 338–339

Reports, generating bug
 Android Debug Bridge, 477
 Google Analytics, 284–285
 Google Analytics Dashboard, 288–290

requestLocationUpdates() method, 254

requestRestore() method, 435

requestStop() method, OpenGL ES thread, 350

requiredAccountType attribute, restricted profiles, 429

/res/animator/grow.xml, animations, 333

/res/animator/resource, animations, 332–333, 337–339

/res/drawable, internationalization, 441

/res/drawable/resource, shape drawables, 315–316

/res/layout, internationalization, 440–441

/res/values, internationalization, 440–441

/res/values/styles.xml, styles, 106–109

/res/xml, live wallpaper, 406

Resolution, bitmap, 314

Restore, forcing, 477

RestoreObserver object, 435

restrictedAccountType attribute, profiles, 429

Restricted profiles, tablets, 428–429

Return values, building NDK project, 381–382

Reverse geocoding, 256

RFCOMM connections, Bluetooth, 235

Right-to-left (RTL) language localization, 445

RingtoneManager object, 208–209

Ringtones, 207–209

Roaming, 214

RotateAnimation class, 335

Rotating transformations, 335

RoundRectShape object, 318–319

RTL (right-to-left) language localization, 445

runOnUiThread() method, Activity, 15

Runtime, L Developer Preview, 460–462

S

sample.html file, 185

Sans Serif typeface, drawing on screen, 310

Saving, game data with Cloud Save, 299

Scale
 bitmaps, 313
 designing flexible user interfaces, 152
 loading content into WebView, 178
 monitoring battery use, 233
 working with transformations, 335–336

ScaleAnimation class, 336

ScaleGestureDetector class, 123, 129–131

Scenes
 lighting 3D, 358–360
 state animations with, 342

Schema
 creating SQLite database, 37–38
 designing SQLite database, 491–492

Schema (*continued*)

- listing for database with `sqlite3`, 488
- listing for table with `sqlite3`, 487

Screens

- 2D drawing on. *See* **Drawing 2D objects**
- 3D drawing on. *See* **Drawing 3D objects**
- designing flexible user interfaces, 152
- handling orientation changes, 134–136
- listening for events on entire, 120–121
- listening for focus changes, 122–123
- listening for long-click event, 121–122
- listening for touch mode changes, 119–120
- optimizing web applications for Google TV, 157
- removing action bars from, 104–105
- using alternative resources for, 153
- using space effectively on big landscape, 153–154
- WebView control as entire, 178

Screen saver, Daydream, 408–410**Scripts, SQL, 489****Scroll gestures, 124–125, 128–129****SDK Manager, Android Studio, 508–509****Search**

- in-application, 416–417
- basic, 417
- configuring Android manifest file for, 423–424
- creating search Activity, 422–423
- global, 424–425
- making application content searchable, 415–416
- multimedia APIs, 207–208
- quiz Q & A, 426, 525
- requesting, 420–421

- suggestions, 417–420
- voice, 420

Search button, deprecated, 420**SearchManager class, 417****searchSuggestAuthority attribute, 418****searchSuggestThreshold attribute, 418****SearchView class, 420–421****Secondary logs, accessing, 476****Security**

- application broadcast, 73–74
- host card emulation applications, 245
- JavaScript control for Android app, 187
- software piracy. *See* **Software piracy protection**
- SSL, 164–166
- video content, with `MediaDrm` class, 202

Seed feature, stress testing applications, 480**SELECT statement, querying SQLite database tables, 493****Semicolon (;), `sqlite3`, 490****sendBroadcast() method, 67****Sending**

- broadcasts, 67–68
- enabling Android Beam, 241–243
- SMS, 218–220

sendOrderedBroadcast() method, 67**sendStickyBroadcast() method, 67****sendStickyOrderedBroadcast() method, 67****Sensor class, 226–227****SensorEvent class, 228****SensorEventListener object, 228****SensorManager class, 226–227****Sensors. *See* Hardware sensors****separator command, `sqlite3`, 489****Sequential tweened animations, 333****Serial number, ADB commands to specific devices, 470**

Serif typeface, drawing on screen, 310

Servers

- alternative to GCM, 275
- integrating GCM on Android application, 274

Service class

- creating App Widget, 393
- creating App Widget update Service, 398–399
- implementing Daydream, 409
- updating App Widget, 398–399
- working with live wallpapers, 404–408

ServiceConnection object, remote interface, 27–28

Service information, requesting, 214

<service> manifest tag

- configuring App Widgets, 399
- configuring live wallpapers, 406–407
- creating Service, 25
- implementing remote interface, 27
- registering Service implementation, 20

Services

- controlling, 25
- creating, 20–25
- default messaging applications, 217
- implementing IMEs as Android, 117
- implementing IntentService class, 30–33
- implementing Parcelable class, 28–30
- implementing remote interface, 26–28
- lifecycle, 20
- overview of, 19
- quiz Q & A, 34, 519
- when to use, 19

Service Set Identifier (SSID), Wi-Fi state, 247

ServiceState object, call state, 213–214

Session Initiation Protocol (SIP), Telephony API, 222–223

<set> tag, tweened animations, 333

setAccuracy() method, location of device, 255

setAutoCancel() method, notifications, 83

setBackground() method, drawable animations, 330

setBeam PushUri() method, Android Beam over Bluetooth, 245

setBitmap() method, wallpaper, 199

setBuiltInZoomControls() method, WebView, 178

setClass() method, broadcasts, 74

setColor() method, Paint, 307

setContentDescription() method, accessibility, 140

setContentText() method, notifications, 80, 87, 89

setContentTitle() method, notifications, 80, 87, 89

setContentView() method, themes, 112

setContentView() method, WebView, 178

setDataSource() method, playing audio, 205

setDisplayHomeAs UpEnabled() method, icon clicks, 104

setEGLContextClient Version() method, OpenGL ES 2.0, 370

setFlags() method, hardware acceleration, 325

setHapticFeedbackEnabled() method, accessibility, 140

setHTMLText() method, web extensions, 185–186

setInitialScale() method, WebView, 178

setInterpolator() method, interpolator, 341

setJavaScriptEnabled() method

web extensions, 183

WebView, 178

setLayerType() method, hardware acceleration, 325

setLightTouchEnabled() method, WebView, 178

setMediaController() method, VideoView, 202–203

setNdefPushMessageCallback() method, Android Beam, 242

setNotificationUri() method, content provider, 57

setOnClickListener() method, 123, 504

setOnCompletionListener() method, video, 202–203

setOneShot() method, drawable animation, 330

setOnFocusChangeListener() method, focus changes, 122–123

setOnLongListener() method, View, 123

SetPackage() methods, application broadcasts, 74

setParameters() method, camera, 196

setPowerRequirement() method, location of device, 255

setPreviewFormat() method, camera, 195

setPrimaryClip() method, copying/pasting, 119

setResource() method, wallpaper, 199

setShader() method, Paint, 307–309

setSound() method, notifications, 86

setStream() method, wallpaper, 199

setStyle() method, notifications, 88–90

setSupportZoom() method, WebView, 178

setTables() method, content provider, 57

setTarget() method, property animations, 339

setTheme(), 111–113

Settings

- Daydream, 409
- WebView, 178–179

Settings, Language & import menu, 115

setTint() method, drawable resources at runtime, 462

setVibrate() method, notifications, 84

setVideoURI() method, video, 202–203

setWebChromeClient() method, 179

SGL (Silicon Graphics), and OpenGL, 345

SHA-1 fingerprint, map API key, 263–264

Shaders, OpenGL ES 2.0, 370–373

Shadows, L Developer Preview, 462–463

ShapeDrawable

- defining as XML resources, 315–316
- defining programmatically, 316–317
- using ArcShape object, 320–322
- using OvalShape object, 319–320
- using PathShape object, 323–324
- using RectShape object, 318
- using RoundRectShape object, 318–319
- using view animations, 331–332

Shapes

- defining shape drawables, 315–317
- drawing arcs, 320–322
- drawing ovals and circles, 319–320
- drawing paths, 322–324
- drawing rectangles and squares, 318
- drawing rectangles with rounded corners, 318–319

Shared

- audio, 206–207
- preference files, backing up, 432
- still images, 198

Shell commands, ADB

- accessing sqlite3 tool, 486
- inspecting SQLite databases, 478
- installing custom binaries via, 481–482
- issuing single shell commands, 471
- starting and stopping emulator, 471–472
- stress testing applications, 478–481
- using shell session, 471

Shorthand, Java, 500–504

- showAsAction attribute, action bar, 102–103
- showVoiceSearchButton value, voice search, 420
- shutdown() method, text-to-speech, 147
- Sidebars, conventions used in this book, 7
- Signal strength, Telephony API, 214–215
- Silicon Graphics (SGI), and OpenGL, 345
- SIM operator name, requesting service information, 214
- SimpleAccessProvider application, 63–64
- SimpleAppWidgetActivity, 397
- SimpleBackup application, 430–435
- SimpleBroadcasts application. See Broadcasts
- SimpleCursorAdapter
 - binding data to controls, 50–53
 - creating search Activity, 423
 - retrieving content provider data, 64
- SimpleDataUpdateService class, App Widgets, 398
- SimpleGestures application, 125–129
- SimpleIntentService application, 31–33
- SimpleInternationalization application. See Internationalizing applications
- SimpleLiveWallpaper application, 406
- SimpleNetworking application, 164–166
- SimpleNotifications application. See Notifications
- SimpleOnScaleGestureListener class, multitouch, 130
- SimpleOpenGL application. See Graphics, 3D applications
- SimpleOrientation application, 135–136
- SimplePropertyAnimation application, 337–341
- Simple queries, 43–44
- SimpleSearchableActivity, 422–423
- SimpleSearchProvider application. See Content providers
- SimpleSpeech application, 141–145
- SimpleTextInputTypes application. See Textual input methods
- SimpleWeb application. See Web APIs
- SimpleWebExtension application, 183–187
- SimpleWireless application, 238–239
- Simultaneous tweened animations, 333
- Single-touch gestures
 - GestureDetector class detecting, 123
 - handling common, 124–129
 - overview of, 123
- SIP (Session Initiation Protocol), Telephony API, 222–223
- Siri speech-recognizing assistant, Apple, 139
- Size
 - calculating App Widget, 394
 - GCM for Android limits, 272
 - SQLite limits, 491
- Smartphones
 - audience for, 153
 - challenges of games for, 154
 - Fragment-based design for, 152
 - scaling graphics for, 154
 - tablets vs., 154
- SmartSmoothZoom() method, camera, 197
- SMS
 - applications other than default, 217–218
 - default messaging application, 215–216
 - overview of, 216
 - permissions to send/receive messages, 218
 - sending, 218–220
- SmsManager, 218–220
- SMS Provider, 217
- SoftKeyboard legacy sample application, 117
- Software developers, 1, 6–7

Software keyboards

- choosing appropriate, 116–117
- customizing, 117–118
- input using, 115
- Speech Recording option, 141–142
- text input, 115

Software piracy protection

- obfuscating with ProGuard, 450–452
- other antipiracy tips, 453–454
- overview of, 449
- quiz Q & A, 454, 525
- secure coding practices, 450
- using License Verification Library, 452–453
- vulnerability of all applications, 449

speak() method, text-to-speech, 147**speech package, 141–145****Speech recognition framework, accessibility, 141–145****SpeechRecognizer class, 141****SQL**

- executing commands on sqlite3, 490
- executing scripts from files with sqlite3, 489

SQLite databases. See also sqlite3**command-line**

- binding data to application user interface, 48–53
- closing and deleting, 45–46
- common tasks, 485
- content providers. *See* **Content providers**
- creating, 36–38
- deleting records, 39–40
- designing persistent databases, 46–48
- inserting records, 38
- inspecting using ADB shell, 478
- limitations of, 490–491

- overview of, 35, 485
- querying, 40–45
- quiz Q & A, 54, 519
- storing structured data, 35–36
- transactions, 40
- updating records, 38–39

SQLite databases, example

- altering/updating data in tables, 495
- calculated columns, 496–497
- creating tables with AUTOINCREMENT, 492
- deleting tables, 497
- designing schema, 491
- foreign/composite primary keys, 493–494
- inserting data into tables, 492–493
- overview of, 491
- querying multiple tables using JOIN, 495
- querying tables for results with SELECT, 493
- quiz Q & A, 498, 526
- setting typeface, 492
- subqueries for calculated columns, 497

SQLite FTS3 extension, 420**sqlite3 command-line**

- connecting to SQLite database, 486–487
- debugging tool for database state, 36
- executing SQL commands, 490
- exploring database, 487–488
- finding application database file on device, 36
- importing/exporting database and its data, 488–489
- inspecting SQLite database via ADB shell, 478
- launching ADB shell, 486

- other commands, 490
 - overview of, 486
 - using ADB shell interface to run, 472
- SQLiteDatabase instance, 36, 37**
- SQLiteOpenHelper class, 46–48**
- SQLiteQueryBuilder, 44, 57–58**
- Squares, drawing, 322–324**
- /src folders, Android Studio project, 512**
- SSID (Service Set Identifier), Wi-Fi state, 247**
- startActivity() method, 221, 263**
- startActivityForResult() method, 145, 237**
- startAngle parameter, arcshape, 321–322**
- start command, emulator, 472**
- startDiscovery() method, Bluetooth devices, 238**
- startDrag() method, drag-and-drop, 134**
- start() method**
- drawable animation, 330
 - property animations, 339
- startOffset property, tweened animation, 333**
- startscan() method, Wi-Fi state, 247**
- start-server command, ADB server, 470**
- startService() method, 20, 25**
- State**
- animations with scenes/transitions, 342
 - managing WebView, 181–182
 - monitoring Wi-Fi, 246–248
 - permission to access information on phone, 212
 - requesting call, 212–214
- Static inner classes, 504**
- Statistics. See Google Analytics**
- Status bar**
- customizing notifications, 86–88
 - displaying notification queue on, 80–82
 - notifications displayed on, 77–78
 - notifying users with, 78–79
 - updating notifications, 81–83
- Status, retrieving Android network, 171–173**
- Stepping through code, Android Studio, 516**
- Sticky broadcasts, 67**
- STILL, activity recognition APIs, 261**
- Still images**
- assigning as wallpaper, 199
 - camera mode settings, 196
 - camera parameters, 197
 - capturing with camera, 192–196
 - choosing device camera, 199–200
 - debugging with Chrome DevTools, 187
 - sharing images, 198
 - during video sessions, 201
 - working with multimedia, 191–192
 - zooming camera, 197
- stop command, emulator, 471**
- stopFaceDetection(), camera, 204**
- stop() method**
- drawable animations, 331
 - playing audio, 205
- stopService() method, 20, 25**
- stopSmooth Zoom() method, camera, 197**
- Storage**
- gathering statistics and avoiding, 292
 - SQLite databases for structured data, 35–36
 - SQLite limitations for procedures, 491
- STREAM_NOTIFICATION, making noise, 86**
- Stretchable graphic formats, 152**
- StrictMode**
- impact on networking code, 164
 - networking code requiring, 12
 - responsiveness of applications, 17
- Structure, of this book, 1–3**

Styles

- L Developer Preview improvements, 462
- leveraging inheritance, 109–111
- notification, 88–90
- Paint, 307, 323–324
- simple, 106–109
- themes, 111–113
- user interface design, 106

<style> tag, 106, 111–113

submitScore() method, leaderboards, 298

Subqueries, SQLite database, 497

supportsRtl attribute, RTL language localization, 445

@SuppressWarnings option, exceptions with native code, 383

surfaceCreated() method

- Camera, 192–195
- starting OpenGL ES thread, 349–350

surfaceDestroyed() method, Camera, 193, 195

SurfaceHolder, 193–195, 202–203

SurfaceHolder.Callback, SurfaceView, 348–349

SurfaceView

- creating for OpenGL ES, 348–349
- enabling OpenGL threads, 362–365
- initializing GLS, 352
- initializing OpenGL ES, 352
- OpenGL ES 2.0, 370–373
- OpenGL ES APIs in Android SDK, 347
- starting OpenGL ES thread, 349–350

sweepAngle parameter, arcshape, 321–322

Sweep gradients, 309

Sync adapters, synchronizing data with, 429–430

Synchronizing data

- not using backup services for, 430
- notifications, L Developer Preview, 463
- overview of, 429–430

Syntax, Java, 500–504

Synthesized speech, text-to-speech, 145

System images, Android TV, 464–465

System-wide accounts, 428

T
Tables, SQLite database

- adding data, 492
- altering/updating data, 495
- creating, 37, 492
- deleting, 46, 497
- dumping contents with sqlite3, 488–489
- foreign/composite primary keys, 493–494
- listing available with sqlite3, 487
- querying, 493
- querying multiple, 495

Tablets

- attracting new types of users, 153
- designing flexible user interface, 152
- developing applications, 154–155
- overview of, 151
- quiz Q & A, 159, 521
- using screen space effectively, 153–154

takePicture() method, Camera class, 195, 201

TalkBack application, accessibility, 140

Target Class, application broadcasts, 74

Telephony APIs

- making phone calls, 220–221
- monitoring signal strength/data speed, 214–215
- permission to access information, 212
- quiz Q & A, 223, 521–522
- receiving phone calls, 221–222
- requesting call state, 212–214
- requesting service information, 214
- using SMS, 216–220
- working with phone numbers, 215–216
- working with SIP, 222–223
- working with telephony utilities, 211–212

TelephonyManager object, 212–214**Temp variables, unnecessary Java, 501****10-foot experience, Google TV, 157****Ternary operations, Java, 503****Testing**

- Android Wear, 158
- applications for ART, 460
- asynchronous code on real devices, 12
- backup services, 435
- custom locales, 442
- Google Play game services, 296
- SQL queries with sqlite3, 490

Text

- drawing on screen, 310–312
- layouts with WebView, 176
- measuring screen for, 312
- prediction, 118

Text notifications

- components, 79
- creating with icon, 79–80
- customizing, 86–88

- displaying on notification queue, 80–82
- expandable and contractible, 88–90
- updating, 83

Text-to-speech (TTS) services, accessibility, 145–147**Textual input methods**

- customizing software keyboards, 117–118
- overview of, 115
- text prediction and user dictionaries, 118
- using clipboard framework, 118–119
- working with software keyboards, 115–117

Texturing objects, 3D graphics, 359–362**Text view, Android Studio, 514–515****TextView control**

- building simple styles, 106–109
- implementing text-to-speech, 147
- parallel execute to update, 14
- updating in UI, 13
- updating with Thread, 16
- working with view animations, 331–332

Themes, user interface design, 111–113**Third-party services**

- in-app billing APIs, 280
- backup services, 430
- push messaging services, 275
- that use ADB for package installation, 473

Thread class

- asynchronous network operations, 168–169
- offload processing off main UI thread, 12
- working with, 15–16

Threading and asynchronous processing

- AsyncTask class, 12–14
- importance of, 11–12
- Loaders, 16
- overview of, 11
- quiz Q & A, 17–18, 519
- StrictMode, 17
- Thread class, 15–16
- ThreadPoolExecutor, 16**
- Throttling mechanism, Google Cloud Messaging, 272**
- ThrowNew() method, exceptions with native code, 382–383**
- Ticker text, 79–82**
- TILTING, activity recognition APIs, 262**
- timeDistanceFactor, single-touch gestures, 129**
- TimeEvaluator class, property animation, 337**
- time mode, LogCat logging, 474**
- tint attribute, drawable resources at runtime, 462**
- Title text, notifications, 79, 90**
- toAlpha value, 334–335**
- Toast messages, 24**
- ToDegrees property, rotating transformations, 335**
- toggleFPSDisplay() method, OpenGL/application threads, 365**
- Tokens, 272, 428**
- Touch mode, listening for changes, 119–120**
- toXDelta, fromYDelta values, 336**
- toXScale, toYScale values, 335–336**
- Tracking ID, Google account for Analytics, 284**
- TransactionBuilder class, 290–292**
- Transactions, SQLite application databases, 40**
- Transformations, animation, 333**

- TransitionManager, 342**
- Transitions, state animations with, 342**
- TranslateAnimation class, 336**
- Translation services, internationalizing applications, 445**
- Transparency, alpha transformations, 334–335**
- Trial editions, preventing application piracy with free, 454**
- Triggers**
 - creating SQLite database, 37–38
 - SQLite limitations, 491
- True north, finding, 230**
- try/catch block, SQLite database transactions, 40**
- TTS (text-to-speech) services, accessibility, 145–147**
- Turn-based multiplayer APIs, Google Play game services, 299**
- TV**
 - Android, 464–465
 - attracting new types of users, 153
 - Google, 155–158
 - overview of, 151
 - quiz Q & A, 159, 521
 - using screen space effectively, 153–154
- Tweened animations. See View (tweened) animations**
- TYPE_ALARM, system events, 208**
- Typefaces, 310–312**
- TYPE_NOTIFICATION, 208**
- TYPE_ORIENTATION sensor value, 230**

U

UI thread

- AsyncTask class, 12–14
- moving network operations off main, 167–171

- offload processing of main, 12
 - starting OpenGL ES, 349–350
 - Thread class, 15–16
- Unary operations, Java, 502**
- unbindService() method, remote interface, 27–28**
- Uninstallation, of applications using ADB, 473**
- uninstall command, 473**
- UNKNOWN, activity recognition APIs, 262**
- updateAppWidget() method, 398**
- update() method**
- building content provider, 59–60
 - records in SQLite database, 38–39
- updatePeriodMillis attribute, App Widgets, 394**
- Updates**
- Android Studio versions, 507
 - antipiracy tips, 453
 - App Widgets, 394–399
 - content provider, 62
 - data in SQLite database tables, 495
 - global search, 425
- UriMatcher class, 57, 58–59, 419**
- URIs**
- building content provider, 56
 - enabling search suggestions, 419
 - locating content, 63–64
- URL object, 165–166, 171**
- URLUtil class, WebKit API, 182**
- USB, 239–241**
- UsbAccessory object, 240**
- UsbManager class, 240–241**
- Use case. See Extending Android application reach**
- User accounts**
- backup service. *See Backup service*
 - managing with Account Manager, 427–428
 - multiple users, restricted profiles and, 428–429
 - overview of, 427
 - synchronizing data, 429–430
- User dictionaries, 118**
- User input methods**
- accessibility with alternative, 140
 - fine-tuned control over, 116
 - gestures. *See Gestures*
 - quiz Q & A, 137, 520
 - screen orientation changes, 134–136
 - speech recognition services, 141–145
 - textual, 115–119
 - user events, 119–123
- User interface**
- action bars. *See Action bars*
 - Android guidelines, 97–98
 - Android Studio, 513–515
 - binding data to application, 48–53
 - challenges of games, 154
 - content provider and, 55–56
 - contextual action mode, 105–106
 - creating devices with flexible, 152
 - defining App Widgets, 396–397
 - device diversity and, 151–154
 - Google TV variations, 156–158
 - overview of, 97
 - quiz Q & A, 113
 - state animations with scenes/transitions, 342
 - style inheritance, 109–111
 - styles, 106–109
 - themes, 111–113

UserDictionary content provider, 118**Users**

- attracting to new types of devices, 153
- designing notifications, 77–79, 91–92
- with disabilities. *See* **Accessible applications**
- L Developer Preview improvements for, 461–464
- protecting privacy when collecting statistics, 293

Users overview reports, Google Analytics, 288**<uses-feature> manifest tag**

- configuring Android Beam, 244–245
- configuring for Open GL ES 2.0, 369
- configuring sensors, 227
- configuring USB, 240–241
- declaring Bluetooth, 237
- declaring device features, 152
- improving performance with Android NDK, 384
- OpenGL ES device compatibility, 346–347
- requesting CAMERA permissions, 192
- using GPS, 254

<uses-sdk> manifest tag

- configuring for Open GL ES 2.0, 384
- creating Android NDK Project, 380
- creating live wallpaper, 407
- improving graphics performance, 380

V

Validation, parsing XML, 167**ValueAnimator class, property animation, 337****valueFrom attribute, property animation, 338–339****valueTo attribute, property animation, 338–339****valueType attribute, property animation, 338****Variables, unnecessary Java temp, 501****Verbose logging, 478****Versions**

- Android SDK OpenGL ES, 346
- Android Studio updates, 507
- devices running Android NDK, 377–378
- preventing application piracy by blocking old, 453–454

Vertex buffer, drawing vertices, 353–355**Vertices**

- coloring, 355–356
- drawing 3D objects, 353–354
- lighting scenes, 358–360

Vibration, phone notifications, 83**Video**

- playing, 202–203
- recording, 200–201

VideoView widget, 202–203**view.accessibility package, 140****View attribute values, 106****View class, 305–306****View controls**

- accessibility, 140
- detecting user motion, 123–124
- handling user events, 119–123
- hardware acceleration, 325–326
- property animations, 339–341
- varying for animation, 331

ViewGroup, scene state transitions, 342**View hierarchies, RemoteViews, 396****View objects**

- contextual action mode, 105
- making styles, 111
- themes, 111–113

Viewport, OpenGL ES, 352**ViewPropertyAnimator class, 337, 340**

Views

interacting with OpenGL ES and Android, 362–365

SQLite limitations, 491

ViewTreeObserver class, 120–121**View (tweened) animations**

alpha transparency transformations, 334–335

defined, 329

defining as XML resources, 332–333

defining programmatically, 333

defining simultaneous/sequential, 333

loading, 334

moving transformations, 336

rotating transformations, 335

scaling transformations, 335–336

working with, 331–332

View widgets, L Developer Preview, 462–463**Voice search, enabling, 420****Vulnerabilities, software piracy protection, 450**

W

Wallpaper

live, 404–408

still images as, 192, 199

WallpaperManager class, 192, 199**WallpaperService class, 404–408****<wallpaper> XML tag, 406****Wearables**

attracting new types of users, 153

developing applications, 158–159

quiz Q & A, 159, 521

Web APIs

browsing with WebView. *See*

WebView control

building web extensions, 182–187

debugging WebViews, 187

overview of, 175

quiz Q & A, 188, 521

working with Adobe AIR and Flash, 187–188

WebBackForwardList class, WebKit API, 182**WebChromeClient class, 179–181, 183–184****WebHistoryItem class, WebKit API, 182****WebKit rendering engine, 175, 182****WebSettings class, WebView, 178–179****WebViewClient class, 179****WebView control**

adding browser chrome, 179–181

adding features, 178

browsing Web, 175–176

building web extensions, 182–187

designing layout, 176

handling events, 179–180

loading content, 176–178

managing state, 181–182

modifying settings, 178–179

WHERE clause

executing simple queries, 43–44

remove() method, 39–40

update() method, 38

Wi-Fi

monitoring state, 246–248

overview of, 245

Wi-Fi Direct, 245–246

WifiManager object, 246–248**WifiP2pManager class, 246****Wildcards, enabling search, 419–420****Work queue, 30–33****World Magnetic Model, 230****writeToParcel() method, Parcelable class, 29****wtf error, filtering log events, 475**

X

XML

- creating App Widget, 393–394
- creating search configuration, 417–420
- defining property animation, 337–339
- defining shape drawables, 315–316
- defining tweened animations, 332–333
- editor, in Android Studio text view, 514–515
- parsing from network, 166–167

XMLPullParser()method, 166–167

XMPP, 275

Z

Zoom

- camera settings, 197
- modifying WebView control, 178