This page intentionally left blank
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.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data is on file.

Copyright © 2012 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.

Android is a trademark of Google, Inc. Pearson Education does not assert any right to the use of the Android trademark, and neither Google nor any other third party having any claim in the Android trademark have sponsored or are affiliated with the creation and development of this book.

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 2.5 Attribution license (http://creativecommons.org/licenses/by/2.5/).


Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing, July 2012
This book is dedicated to ESC.
This page intentionally left blank
Contents

Introduction 1

I: Advanced Android Application Design Principles

1 Threading and Asynchronous Processing 9
   The Importance of Processing Asynchronously 9
   Working with the AsyncTask Class 10
   Working with the Thread Class 13
   Working with Loaders 14
   Understanding StrictMode 14
   Summary 15
   References and More Information 15

2 Working with Services 17
   Determining When to Use Services 17
   Understanding the Service Lifecycle 18
   Creating a Service 18
   Controlling a Service 23
   Implementing a Remote Interface 24
   Implementing a Parcelable Class 26
   Using the IntentService Class 29
   Summary 33
   References and More Information 33

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
6 Working with Notifications 83
Notifying the User 83
A Word on Compatibility 84
Notifying with the Status Bar 84
Using the NotificationManager Service 85
Creating a Simple Text Notification with an Icon 85
Working with the Notification Queue 86
Updating Notifications 88
Clearing Notifications 90
Vibrating the Phone 91
Blinking the Lights 92
Making Noise 93
Customizing the Notification 94
Designing Useful Notifications 96
Summary 97
References and More Information 97

II: Advanced Android User Interface Design Principles

7 Designing Powerful User Interfaces 99
Following Android User Interface Guidelines 99
Working with Menus 100
Using Options Menus 100
Using Context Menus 103
Using Popup Menus 105
Enabling Action Bars 105
Building Basic Action Bars 106
Customizing Your Action Bar 110
Handling Application Icon Clicks on the Action Bar 112
Working with Screens That Do Not Require Action Bars 114
Introducing Contextual Action Mode 114
Using Advanced Action Bar Features 114
8 Handling Advanced User Input

Working with Textual Input Methods 123
   Working with Software Keyboards 123
   Working with Text Prediction and User Dictionaries 126
   Using the Clipboard Framework 126
Handling User Events 127
   Listening for Touch Mode Changes 127
   Listening for Events on the Entire Screen 128
   Listening for Long Clicks 129
   Listening for Focus Changes 130
Working with Gestures 131
   Detecting User Motions Within a View 131
   Handling Common Single-Touch Gestures 132
   Handling Common Multi-Touch Gestures 139
   Making Gestures Look Natural 142
   Using the Drag and Drop Framework 143
Handling Screen Orientation Changes 144
Summary 146
   References and More Information 147

9 Designing Accessible Applications

Exploring the Accessibility Framework 149
   Leveraging Speech Recognition Services 151
   Leveraging Text-To-Speech Services 155
Summary 158
   References and More Information 158
10 Best Practices for Tablet and Google TV Development 159

Understanding Device Diversity 159
  Don’t Make Assumptions about Device Characteristics 159
  Designing Flexible User Interfaces 160
  Attracting New Types of Users 161
  Leveraging Alternative Resources 161
  Using Screen Space Effectively on Big Landscape Screens 161
Developing Applications for Tablets 162
Developing Applications for Google TV 164
  Optimizing Web Applications for Google TV 165
  Developing Native Android Applications for Google TV 165
Developing Apps for the Amazon Kindle Fire 166
Summary 167
  References and More Information 168

III: Leveraging Common Android APIs

11 Using Android Networking APIs 169

Understanding Mobile Networking Fundamentals 169
  Understanding Strict Mode with Networking 170
Accessing the Internet (HTTP) 170
  Reading Data from the Web 170
  Using HttpURLConnection 171
Parsing XML from the Network 172
Handling Network Operations Asynchronously 174
  Retrieving Android Network Status 179
Summary 181
  References and More Information 181

12 Using Android Web APIs 183

Browsing the Web with WebView 183
  Designing a Layout with a WebView Control 184
  Loading Content into a WebView Control 184
# Contents

13 Using Location-Based Services APIs  
- Using Global Positioning Services (GPS) 197  
  - Using GPS Features in Your Applications 198  
  - Determining the Location of the Device 198  
  - Locating Your Emulator 200  
- Geocoding Locations 200  
  - Mapping Locations 204  
    - Mapping Intents 205  
    - Mapping Views 206  
    - Getting Your Debug API Key 207  
    - Panning the Map View 209  
    - Zooming the Map View 210  
    - Marking the Spot 211  
- Doing More with Location-Based Services 216  
  - Summary 217  
    - References and More Information 217

14 Using Android Multimedia APIs  
- Working with Multimedia 219  
- Working with the Camera 220  
  - Capturing Still Images Using the Camera 220  
- Working with Video 229  
- Working with Face Detection 233
17 Using Android's Optional Hardware APIs 263

Working with Bluetooth 263
  Checking for the Existence of Bluetooth Hardware 264
  Enabling Bluetooth 264
  Querying for Paired Devices 265
  Discovering Devices 265
  Establishing Connections Between Devices 266

Working with USB 267
  Working with USB Accessories 268
  Working as a USB Host 269

Working with Android Beam 269
  Enabling Android Beam Sending 270
  Receiving Android Beam Messages 271
  Configuring the Manifest File for Android Beam 272

Working with Wi-Fi 273
  Introducing Wi-Fi Direct 273
  Monitoring Wi-Fi State 274

Summary 276
  References and More Information 276

IV: Drawing, Animations, and Graphics Programming with Android

18 Developing Android 2D Graphics Applications 279

Drawing on the Screen 279
  Working with Canvases and Paints 279
21 Using the Android NDK 355
Determining When to Use the Android NDK 355
Installing the Android NDK 356
Exploring the Android NDK 357
    Running an Android NDK Sample Application 357
Creating Your Own NDK Project 357
    Calling Native Code from Java 358
    Handling Parameters and Return Values 359
    Using Exceptions with Native Code 360
    Using Native Activities 362
Improving Graphics Performance 362
A Comparison to RenderScript 363
Summary 363
    References and More Information 364
V: Maximizing Android's Unique Features

22 Extending Android Application Reach 365
   Enhancing Your Applications 365
   Working with App Widgets 366
      Creating an App Widget 367
      Installing an App Widget 374
      Becoming an App Widget Host 375
   Working with Live Wallpapers 375
      Creating a Live Wallpaper 376
      Creating a Live Wallpaper Service 376
      Creating a Live Wallpaper Configuration 378
      Configuring the Android Manifest File for Live Wallpapers 379
      Installing a Live Wallpaper 379
   Acting as a Content Type Handler 381
   Determining Intent Actions and MIME Types 382
      Implementing the Activity to Process the Intents 383
      Registering the Intent Filter 384
   Summary 384
      References and More Information 384

23 Enabling Application Search 385
   Making Application Content Searchable 385
      Enabling Searches in Your Application 386
      Enabling Global Search 395
   Summary 398
      References and More Information 398

24 Working with Cloud to Device Messaging 399
   An Overview of C2DM 399
      Understanding C2DM Message Flow 400
      Understanding the Limitations of the C2DM Service 400
   Signing Up for C2DM 401
   Incorporating C2DM into Your Applications 402
   Exploring the C2DM Sample Applications 403
28 Enabling Application Statistics with Google Analytics 429
Creating a Google Account for Analytics 429
Adding the Library to Your Eclipse Project 431
Collecting Data from Your Applications 432
Logging Different Events 432
Using the Google Analytics Dashboard 433
Gathering eCommerce Information 436
Logging eCommerce Events in Your Applications 436
Reviewing eCommerce Reports 437
Tracking Ad and Market Referrals 438
Gathering Statistics 438
Protecting Users’ Privacy 439
Summary 439
References and More Information 439

29 Protecting Applications from Software Piracy 441
All Applications Are Vulnerable 441
Using Secure Coding Practices 442
Obfuscating with ProGuard 442
Configuring ProGuard for Your Android Applications 443
Dealing with Error Reports After Obfuscation 444
Leveraging the License Verification Library 444
Other Anti-Piracy Tips 445
Summary 446
References and More Information 446

VII: Appendices

A The Android Debug Bridge Quick-Start Guide 447
Listing Connected Devices and Emulators 447
Directing ADB Commands to Specific Devices 448
Starting and Stopping the ADB Server 448
Stopping the ADB Server Process 448
Starting and Checking the ADB Server Process 448
B  The SQLite Quick-Start Guide  463
Exploring Common Tasks with SQLite  463
Using the sqlite3 Command-Line Interface  464
  Launching the ADB Shell  464
  Connecting to a SQLite Database  464
  Exploring Your Database  465
  Importing and Exporting the Database and Its Data  466
  Executing SQL Commands on the Command Line  468
  Using Other sqlite3 Commands  469
Understanding SQLite Limitations  469
Learning by Example: A Student Grade Database  469
  Designing the Student Grade Database Schema  470
  Creating Simple Tables with AUTOINCREMENT  470
  Inserting Data into Tables  471
  Querying Tables for Results with SELECT  471
  Using Foreign Keys and Composite Primary Keys  472
  Altering and Updating Data in Tables  473
  Querying Multiple Tables Using JOIN  474
  Using Calculated Columns  474
  Using Subqueries for Calculated Columns  476
Deleting Tables  476

Index  477
Acknowledgments

This book would never have been written without the guidance and encouragement we received from a number of supportive individuals, including our editorial team, coworkers, friends, and family. We’d like to thank the Android developer community, Google, and the Open Handset Alliance for their vision and expertise. Throughout this project, our editorial team at Pearson Education (Addison-Wesley) always had the right mix of professionalism and encouragement. Thanks especially to Trina MacDonald and Laura Lewin, Olivia Basegio, Songlin Qiu, and our crack team of technical reviewers: Doug Jones, Ray Rischpater, and Tony Hillerson, (as well as Dan Galpin, Tony Hillerson, Ronan Schwarz, Charles Stearns, Mike Wallace, and Mark Gjoel, who reviewed previous editions and incarnations of this book). Dan Galpin also graciously provided the clever Android graphics used for tips, notes, and warnings. Amy Badger must be commended for her wonderful waterfall illustration, and we also thank Hans Bodlaender for letting us use the nifty chess font he developed as a hobby project.
About the Authors

Lauren Darcey is responsible for the technical leadership and direction of a small software company specializing in mobile technologies, including Android, Apple iOS, Blackberry, Palm Pre, BREW, J2ME, and 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 B.S. in computer science from the University of California, Santa Cruz.

She spends her free time traveling the world with her geeky mobile-minded husband and 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, drank her way through the beer halls of Germany, slept in the crumbling castles of Europe, and had her tongue stuck to an iceberg in Iceland (while being watched by a herd of suspicious wild reindeer).

Shane Conder has extensive development experience and has focused his attention on mobile and embedded development for the past 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 mobile industry and evaluated mobile development platforms on his tech blogs. He is well-known within the blogosphere. Shane received a B.S. in computer science from the University of California.

A self-admitted gadget freak, Shane always has the latest smartphone, tablet, or other mobile device. He can often be found fiddling with the latest technologies, such as cloud services and mobile platforms, and other exciting, state-of-the-art technologies that activate the creative part of his brain. He is a very hands-on geek dad. He also enjoys traveling the world with his geeky wife, even if she did make him dive with 4-meter-long great white sharks and almost got him eaten by a lion in Kenya. He admits that he has to take at least two phones with him when backpacking—even though there is no coverage—and that he snickered and whipped out his Android phone to take a picture when Laurie got her tongue stuck to that iceberg in Iceland, and that he is catching on that he should be writing his own bio.

The authors have also published several other Android books, including Android Wireless Application Development, Android Wireless Application Development Volume I: Android Essentials, Sams Teach Yourself Android Application Development, Learning Android™ Application Programming for the Kindle Fire™, and the mini-book Introducing Android Development with Ice Cream Sandwich. Lauren and Shane have also published numerous articles on mobile software development for magazines, technical journals, and online
publishers of educational content. You can find dozens of samples of their work in *Linux User and Developer, Smart Developer* magazine (Linux New Media), developer.com, Network World, Envato (MobileTuts+ and CodeCanyon), and InformIT, among others. They also publish articles of interest to their readers at their own Android website, http://androidbook.blogspot.com. You can find a full list of the authors’ publications at http://goo.gl/f0Vlj.
Introduction

Pioneered by the Open Handset Alliance and Google, Android is a popular, free, open-source mobile platform that has taken the wireless world by storm. This book and *Android Wireless Application Development Volume I: Android Essentials* 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 several 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 of the mobile development market today.

- **Quality assurance personnel tasked with testing Android applications.** Whether they are black box or white box testing, quality assurance engineers can find this book invaluable. We devote several chapters to mobile QA concerns, including topics such as developing solid test plans and defect tracking systems for mobile applications, how to manage handsets, and how to test applications thoroughly using all the Android tools available.

- **Project managers planning and managing Android development teams.** Managers can use this book to help plan, hire, and execute Android projects from start to finish. We cover project risk management and how to keep Android projects running smoothly.
• **Other audiences.** This book is useful not only to a software developer, but also for the corporation looking at potential vertical market applications, the entrepreneur thinking about a cool phone application, and the 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 provided in this book to use for fun and profit.

**Why Two Volumes in the Third Edition?**

We wrote the first edition of this book before the Android SDK was released. Now, three years and 14 Android SDK releases later, there is so much to talk about that we’ve had to divide the content of the Android wireless application development process into two separate volumes for this, the third edition.

*Android Wireless Application Development Volume I: Android Essentials* focuses on Android essentials, including setting up your development environment, understanding the application lifecycle and the user interface design, developing for different types of devices, and understanding the mobile software process from design and development to testing and publication of commercial-grade applications.

*Android Wireless Application Development Volume II: Advanced Topics* focuses on advanced Android topics, including leveraging various Android 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 learn to design advanced user interface components and integrate their applications deeply into the platform. Finally, developers learn how to extend their applications beyond traditional boundaries using optional features of the Android platform, including the Android Native Development Kit (NDK), Cloud-To-Device Messaging service (C2DM), Android Market In-Application Billing APIs, Google Analytics APIs, and more.

*Android Wireless Application Development Volume II: Advanced Topics* is divided into seven parts. Here is an overview of the various parts in this book:

- **Part I: Advanced Android Application Design Principles**
  Part I picks up where *Android Wireless Application Development Volume I: Android Essentials* 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, and 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 SDK, including working with action bars and menus, gathering input through nonstandard methods such as gestures and
voice recognition, and much more. You 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, location-based services, multimedia (including the camera), telephony, and hardware sensors.

- **Part IV: Drawing, Animations, and Graphics Programming with Android**
  
  Part IV is for those developers incorporating graphics of any kind into their applications. We cover both 2D and 3D graphics (OpenGL ES and RenderScript), animation, and the Android NDK.

- **Part V: Maximizing Android's Unique Features**
  
  Part V 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 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, leveraging cloud-based services, and backups are just some of the topics discussed.

- **Part VI: Advanced Topics in Application Publication and Distribution**
  
  Part VI covers some more specialized topics in the realm of application publication and distribution, including how to internationalize your applications, enable In-App billing with the Android Market, track application usage patterns with Google Analytics, and take measures to protect your intellectual property from software pirates.

- **Part VII: Appendixes**
  
  Part VII includes a helpful quick start guide for the Android Debug Bridge tool and a refresher course on using SQLite.

### Key Questions Answered in Volume II

This volume of the 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 or RenderScript?

7. What are some of the most commonly used APIs for networking, location-based services, 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 the Android Market’s In-App billing and license verification libraries, Google’s Analytics, and Cloud-to-Device Messaging (C2DM) services?

**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. One Android device became available shortly after we started writing, and it was available only in the United States. Today there are hundreds of devices shipping all over the world—smartphones, tablets, e-book readers, wrist watches, and specialty devices such as the Google TV. 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.

In this new edition, we took the opportunity to do a serious overhaul on book content—but don’t worry, it’s still the book readers loved the first (and second!) time, just bigger, better, and more comprehensive. To cover more of the exciting topics available to Android developers, we had to divide the book into two volumes. In addition to adding tons of 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 backwards compatible. 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 to this edition:
• Coverage of the latest and greatest Android tools and utilities.
• Updates to all existing chapters, often with entirely new sections.
• New chapters, which cover new SDK features or expand upon those covered in previous editions.
• Updated sample code and applications, conveniently organized by chapter.
• Topics such as threading and asynchronous processing, creating content providers, broadcast intents, and animation frameworks now have their own chapters.
• Coverage of hot topics such as tablet and TV design, best practices, Renderscript, in-app billing, and Google Analytics.
• Even more tips and tricks from the trenches to help you design, develop, and test applications for different device targets, including an all-new chapter on tackling compatibility issues.

As you can see, we cover many of the hottest and most exciting features that Android has to offer. We didn’t take this review lightly; we touched every existing chapter, updated content, and added many new chapters. 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 and Mac OS X 10.7.x
• Eclipse Java IDE Version 3.7 (Indigo)
• Eclipse JDT plug-in and Web Tools Platform (WTP)
• Java SE Development Kit (JDK) 6 Update 26
• Android SDK Version 2.3.4, API Level 10 (Gingerbread MR1), Android SDK Version 3.2, API Level 13 (Honeycomb MR2), Android SDK Version 4.0.3, API Level 15 (Ice Cream Sandwich MR1)
  1. ADT plug-in for Eclipse 16.0.1
  2. SDK Tools Revision 16
  3. Android Support Package r4
  4. Android NDK r7
• Android devices: Samsung Galaxy Nexus, Motorola Droid 3, Samsung Galaxy tab 10.1, Asus Transformer Prime, Motorola Atrix 4G, and Logitech Revue
The Android platform continues to aggressively 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, with new editions of the Android platform appearing all the time. Developers can no longer ignore Android as a target platform if they want to reach the smartphone (or smart-device) users of today and tomorrow.

Android’s latest major platform update, Android 4.0, frequently called by its code-name, Ice Cream Sandwich or just ICS, merges the smartphone-centric Android 2.3.x (Gingerbread) and the tablet-centric Android 3.x (Honeycomb) platform editions into a single SDK for all smart-devices, be they phones, tablets, televisions, or toasters. 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. This 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, only a small percentage (less than 5 percent) of users’ devices run Android 3.0 or 4.0. 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.

So what does this mean for this book? It means we provide both 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 that accompanies this book is available for download on the publisher website: http://www.informit.com/title/9780321813848. The source code is also available for download from our book website: http://androidbook.blogspot.com/p/book-code-downloads.html (http://goo.gl/kyAsN). You can also find a variety of Android topics discussed at our book website (http://androidbook.blogspot.com). For example, we present reader feedback, questions, and additional information. You can also find links to our various technical articles on our book website.

**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/
· **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://www.openhandsetalliance.com/

· **Android Market**: Buy and sell Android applications:
  http://www.android.com/market/

· **Mobiletuts+**: Mobile development tutorials, including Android:
  http://mobile.tutsplus.com/category/tutorials/android/

· **anddev.org**: An Android developer forum:
  http://www.anddev.org

· **Google Team Android Apps**: Open source Android applications:
  http://apps-for-android.googlecode.com/

· **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://www.fiercedeveloper.com/

· **Wireless Developer Network**: Daily news on the wireless industry:
  http://www.wirelessdevnet.com/

· **XDA-Developers Android Forum**: From general development to ROMs:

· **Developer.com**: A developer-oriented site with mobile articles:
  http://www.developer.com/

**Conventions Used in This Book**

This book uses the following conventions:

- ➔ is used to signify to readers that the authors meant for the continued code to appear on the same line. No indenting should be done on the continued line.

- Code or programming terms are set in **monospace** text.

- Java import statements, exception handling, and error checking are often removed from printed code samples for clarity and to keep the book a reasonable length.
This book also presents information in the following sidebars:

<table>
<thead>
<tr>
<th>Tip</th>
<th>Tips provide useful information or hints related to the current text.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note</td>
<td>Notes provide additional information that might be interesting or relevant.</td>
</tr>
<tr>
<td>Warning</td>
<td>Warnings provide hints or tips about pitfalls that may be encountered and how to avoid them.</td>
</tr>
</tbody>
</table>

**Contacting the Authors**

We welcome your comments, questions, and feedback. We invite you to visit our blog at:
http://androidbook.blogspot.com
Or, email us at:
androidwirelessdev+awad3ev2@gmail.com
Circle us on Google+:
- Lauren Darcey: http://goo.gl/P3RGo
- Shane Conder: http://goo.gl/BpVJh
Leveraging SQLite Application Databases

Applications use a combination of application preferences, the file system, and database support to store information. In this chapter, we explore one of the most powerful ways you can store, manage, and share application data with Android: an application database powered by SQLite. Application databases provide structured data storage that is quick to access, search, and manipulate.

Note
For more information about designing SQLite databases and interacting with them via the sqlite3 command-line tool, please see Appendix B, “The SQLite Quick-Start Guide.” This appendix is divided into two parts. The first half is an overview of the most commonly used features of the sqlite3 command-line interface and the limitations of SQLite compared to other flavors of SQL; the second half of the appendix includes a fully functional tutorial in which you build a SQLite database from the ground up and then use it. If you are new to SQLite or a bit rusty on your syntax, this appendix is for you.

Storing Structured Data Using SQLite Databases
When your application requires a more robust data storage mechanism, you’ll be happy to hear that the Android file system includes support for application-specific relational databases using SQLite. SQLite databases are lightweight and file-based, making them ideally suited for embedded devices.

Tip
Many of the code examples provided in this section are taken from the SimpleDatabase application. This source code for the SimpleDatabase application is provided for download on the book’s websites.
These databases and the data in them are private to the application. To share application data with other applications, you must expose the data you want to share by making your application a content provider.

The Android SDK includes a number of useful SQLite database management classes. Many of these classes are found in the android.database.sqlite package. Here you can find utility classes for managing database creation and versioning, database management, and query builder helper classes to help you format proper SQL statements and queries. The package also includes specialized Cursor objects for iterating query results. You can also find all the specialized exceptions associated with SQLite.

In this chapter, we focus on creating databases in our Android applications. For that, we use the built-in SQLite support to programmatically create and use a SQLite database to store application information. However, if your application works with a different sort of database, you can also find more generic database classes (in the android.database package) to help you work with data from other providers.

In addition to programmatically creating and using SQLite databases, developers can also interact directly with their application’s database using the sqlite3 command-line tool that’s accessible through the ADB shell interface. This can be a helpful debugging tool for developers and quality assurance personnel who might want to manage the database state (and content) for testing purposes.

**Creating a SQLite Database**

You can create a SQLite database for your Android application in several ways. To illustrate how to create and use a simple SQLite database, let’s create an Android project called SimpleDatabase.

**Creating a SQLite Database Instance Using the Application Context**

The simplest way to create a new SQLiteDatabase instance for your application is to use the openOrCreateDatabase() method of your application Context, like this:

```java
import android.database.sqlite.SQLiteDatabase;
...
SQLiteDatabase mDatabase;
mDatabase = openOrCreateDatabase(  
    "my_sqlite_database.db",  
    SQLiteDatabase.CREATE_IF_NECESSARY,  
    null);
```

**Finding the Application Database File on the Device File System**

Android applications store their databases (SQLite or otherwise) in a special application directory:

`/data/data/<application package name>/databases/<database name>`
So, in this case, the path to the database would be
/data/data/com.androidbook.SimpleDatabase/databases/my_sqlite_database.db

You can access your database using the sqlite3 command-line interface using this path.

**Configuring the SQLite Database Properties**

Now that you have a valid SQLiteDatabase instance, it’s time to configure it. Some important database configuration options include version, locale, and the thread-safe locking feature:

```java
import java.util.Locale;
...
```

```java
mDatabase.setLocale(Locale.getDefault());
mDatabase.setLockingEnabled(true);
mDatabase.setVersion(1);
```

**Creating Tables and Other SQLite Schema Objects**

Creating tables and other SQLite schema objects is as simple as forming proper SQLite statements and executing them. The following is a valid `CREATE TABLE` SQL statement. This statement creates a table called `tbl_authors`. The table has three fields: a unique `id` number, which auto-increments with each record and acts as our primary key, and `firstname` and `lastname` text fields:

```sql
CREATE TABLE tbl_authors {
id INTEGER PRIMARY KEY AUTOINCREMENT,
firstname TEXT,
lastname TEXT;
}
```

You can encapsulate this `CREATE TABLE` SQL statement in a static final `String` variable (called `CREATE_AUTHOR_TABLE`) and then execute it on your database using the `execSQL()` method:

```java
mDatabase.execSQL(CREATE_AUTHOR_TABLE);
```

The `execSQL()` method works for nonqueries. You can use it to execute any valid SQLite SQL statement. For example, you can use it to create, update, and delete tables, views, triggers, and other common SQL objects. In our application, we add another table called `tbl_books`. The schema for `tbl_books` looks like this:

```sql
CREATE TABLE tbl_books {
id INTEGER PRIMARY KEY AUTOINCREMENT,
title TEXT,
dateadded DATE,
authorid INTEGER NOT NULL CONSTRAINT authorid REFERENCES tbl_authors(id) ON DELETE CASCADE;
}
```
Unfortunately, SQLite does not enforce foreign key constraints. Instead, we must enforce them ourselves using custom SQL triggers. So we create triggers, such as this one that enforces that books have valid authors:

```java
private static final String CREATE_TRIGGER_ADD =
"CREATE TRIGGER fk_insert_book
BEFORE INSERT ON tbl_books
FOR EACH ROW
BEGIN
SELECT RAISE(ROLLBACK, 'insert on table "tbl_books" violates foreign key constraint "fk_authorid"')
WHERE (SELECT id FROM tbl_authors WHERE id = NEW.authorid) IS NULL;
END;"
```

We can then create the trigger simply by executing the CREATE TRIGGER SQL statement:

`mDatabase.execSQL(CREATE_TRIGGER_ADD);`

We need to add several more triggers to help enforce our link between the author and book tables, one for updating `tbl_books` and one for deleting records from `tbl_authors`.

### Creating, Updating, and Deleting Database Records

Now that we have a database set up, we need to create some data. The `SQLiteDatabase` class includes three convenience methods to do that. They are, as you might expect, `insert()`, `update()`, and `delete()`.

#### Inserting Records

We use the `insert()` method to add new data to our tables. We use the `ContentValues` object to pair the column names to the column values for the record we want to insert. For example, here we insert a record into `tbl_authors` for J.K. Rowling:

```java
import android.content.ContentValues;
...
ContentValues values = new ContentValues();
values.put("firstname", "J.K.");
values.put("lastname", "Rowling");
long newAuthorID = mDatabase.insert("tbl_authors", null, values);
```

The `insert()` method returns the identifier of the newly created record. We use this author identifier to create book records for this author.
Tip
There is also another helpful method called `insertOrThrow()`, which does the same thing as the `insert()` method but throws a `SQLException` on failure, which can be helpful, especially if your inserts do not seem to be working and you’d like to know why. Generally, you’ll want to check values before inserting and not rely on exceptions for common constraints.

You might want to create simple classes (that is, class `Author` and class `Book`) to encapsulate your application record data when it is used programmatically.

Updating Records
You can modify records in the database using the `update()` method. The `update()` method takes four arguments:

- The table to update records
- A `ContentValues` object with the modified fields to update
- An optional `WHERE` clause, in which `?` identifies a `WHERE` clause argument
- An array of `WHERE` clause arguments, each of which is substituted in place of the `?`s from the second parameter

Passing `null` to the `WHERE` clause modifies all records within the table, which can be useful for making sweeping changes to your database.

Most of the time, we want to modify individual records by their unique identifier. The following function takes two parameters: an updated book title and a `bookId`. We find the record in the table called `tbl_books` that corresponds with the `id` and update that book’s title. Again, we use the `ContentValues` object to bind our column names to our data values:

```java
public void updateBookTitle(Integer bookId, String newtitle) {
    ContentValues values = new ContentValues();
    values.put("title", newtitle);
    mDatabase.update("tbl_books",
        values, "id=?", new String[] { bookId.toString() });
}
```

Because we are not updating the other fields, we do not need to include them in the `ContentValues` object. We include only the `title` field because it is the only field we change.

Deleting Records
You can remove records from the database using the `remove()` method. The `remove()` method takes three arguments:

- The table to delete the record from
- An optional `WHERE` clause, in which `?` identifies a `WHERE` clause argument
An array of \texttt{WHERE} clause arguments, each of which is substituted in place of the \texttt{?}s from the second parameter.

Passing \texttt{null} to the \texttt{WHERE} clause deletes all records in the table. For example, this function call deletes all records in the table called \texttt{tbl_authors}:

\begin{verbatim}
mDatabase.delete("tbl_authors", null, null);
\end{verbatim}

Most of the time, though, we want to delete individual records by their unique identifiers. The following function takes a parameter \texttt{bookId} and deletes the record corresponding to that unique \texttt{id} (primary key) in the table called \texttt{tbl_books}:

\begin{verbatim}
public void deleteBook(Integer bookId) {
    mDatabase.delete("tbl_books", "id=?",
                 new String[] { bookId.toString() });
}
\end{verbatim}

You need not use the primary key (\texttt{id}) to delete records; the \texttt{WHERE} clause is entirely up to you. For instance, the following function deletes all book records in the table \texttt{tbl_books} for a given author by the author's unique identifier:

\begin{verbatim}
public void deleteBooksByAuthor(Integer authorID) {
    int numBooksDeleted = mDatabase.delete("tbl_books", "authorid=?",
                                        new String[] { authorID.toString() });
}
\end{verbatim}

\section*{Working with Transactions}

Often you have multiple database operations you want to happen all together or not at all. You can use SQL transactions to group operations together; if any of the operations fail, you can handle the error and either recover or roll back all operations. If the operations all succeed, you can then commit them. Here we have the basic structure for a transaction:

\begin{verbatim}
mDatabase.beginTransaction();
try {
    // Insert some records, update others, delete a few.
    // Do whatever you need to do as a unit, then commit it.

    mDatabase.setTransactionSuccessful();
} catch (Exception e) {
    // Transaction failed. Failed! Do something here.
    // It's up to you.
} finally {
    mDatabase.endTransaction();
}
\end{verbatim}
Now let's look at the transaction in a bit more detail. A transaction always begins with a call to `beginTransaction()` method and a `try/catch` block. If your operations are successful, you can commit your changes with a call to the `setTransactionSuccessful()` method. If you do not call this method, all your operations are rolled back and not committed. Finally, you end your transaction by calling `endTransaction()` in the finally clause, guaranteeing that it'll be called. It's as simple as that.

In some cases, you might recover from an exception and continue with the transaction. For example, if you have an exception for a read-only database, you can open the database and retry your operations.

Finally, note that transactions can be nested, with the outer transaction either committing or rolling back all inner transactions.

### Querying SQLite Databases

Databases are great for storing data in any number of ways, but retrieving the data you want is what makes databases powerful. This is partly a matter of designing an appropriate database schema and partly achieved by crafting SQL queries, most of which are `SELECT` statements.

Android provides many ways in which you can query your application database. You can run raw SQL query statements (strings), use a number of different SQL statement builder utility classes to generate proper query statements from the ground up, and bind specific user interface controls such as container views to your backend database directly.

### Working with Cursors

When results are returned from a SQL query, you often access them using a `Cursor` found in the `android.database.Cursor` class. `Cursor` objects are like file pointers; they allow random access to query results.

You can think of query results as a table, in which each row corresponds to a returned record. The `Cursor` object includes helpful methods for determining how many results were returned by the query the `Cursor` represents and methods for determining the column names (fields) for each returned record. The columns in the query results are defined by the query, not necessarily by the database columns. These might include calculated columns, column aliases, and composite columns.

`Cursor` objects are generally kept around for a time. If you do something simple (such as get a count of records or in cases when you know you retrieved only a single simple record), you can execute your query and quickly extract what you need; don't forget to close the `Cursor` when you're done, as shown here:

```java
// SIMPLE QUERY: select * from tbl_books
Cursor c = mDatabase.query("tbl_books",null,null,null,null,null,null);
// Do something quick with the Cursor here...
c.close();
```
Managing Cursors as Part of the Application Lifecycle

When a Cursor returns multiple records, or you do something more intensive, you need to consider running this operation on a thread separate from the UI thread. You also need to manage your Cursor.

Cursor objects must be managed as part of the application lifecycle. When the application pauses or shuts down, the Cursor must be deactivated with a call to the deactivate() method, and when the application restarts, the Cursor should refresh its data using the requery() method. When the Cursor is no longer needed, a call to close() must be made to release its resources.

As the developer, you can handle this by implementing Cursor management calls within the various lifecycle callbacks, such as onPause(), onResume(), and onDestroy(). If you're lazy, like us, and you don't want to bother handling these lifecycle events, you can hand off the responsibility of managing Cursor objects to the parent Activity by using the Activity method called startManagingCursor(). The Activity handles the rest, deactivating and reactivating the Cursor as necessary and destroying the Cursor when the Activity is destroyed. You can always begin manually managing the Cursor object again later by simply calling stopManagingCursor().

Here we perform the same simple query and then hand over Cursor management to the parent Activity:

```java
// SIMPLE QUERY: select * from tbl_books
Cursor c = mDatabase.query("tbl_books",null,null,null,null,null,null);
startManagingCursor(c);
```

Note that, generally, the managed Cursor object is a member variable of the class, in terms of scope. You may notice that the startManagingCursor() and stopManagingCursor() calls are deprecated. In the context of using data on Android, most databases are exposed as content providers. Using a content provider, one can perform queries similar to these, but on more abstract URIs rather than directly on a database using table names. In doing this, you use the higher-level query() method of the ContentResolver class rather than directly on the database. The proper current method of doing this in a managed way is through the use of the CursorLoader class (android.content.CursorLoader for API Level 11 and higher, and in the support package for API Level 4 and higher).

Iterating Rows of Query Results and Extracting Specific Data

You can use the Cursor to iterate those results, one row at a time using various navigation methods such as moveToFirst(), moveToNext(), and isAfterLast().

On a specific row, you can use the Cursor to extract the data for a given column in the query results. Because SQLite is not strongly typed, you can always pull fields out as Strings using the getString() method, but you can also use the type-appropriate extraction utility function to enforce type safety in your application.
For example, the following method takes a valid Cursor object, prints the number of returned results, and then prints some column information (name and number of columns). Next, it iterates through the query results, printing each record.

```java
public void logCursorInfo(Cursor c) {
    Log.i(DEBUG_TAG, "*** Cursor Begin *** ")
    c.getCount() + ").
    c.getColumnIndex();
    for (int i = 0; i < c.getColumnCount(); i++) {
        rowHeaders = rowHeaders.concat(c.getColumnName(i) + " || ");
    }
    Log.i(DEBUG_TAG, "COLUMNS " + rowHeaders);
    // Print records
    c.moveToFirst();
    while (!c.isAfterLast()) {
        String rowResults = "|| ";
        for (int i = 0; i < c.getColumnCount(); i++) {
            rowResults = rowResults.concat(c.getString(i) + " || ");
        }
        Log.i(DEBUG_TAG, "Row "+ c.getPosition() + ": " + rowResults);
        c.moveToNext();
    }
    Log.i(DEBUG_TAG, "*** Cursor End ***");
}
```

The output to the LogCat for this function might look something like Figure 3.1.

![Figure 3.1](image-url)

**Figure 3.1** Sample log output for the `logCursorInfo()` method.
Executing Simple Queries

Your first stop for database queries should be the query() methods available in the SQLiteDatabase class. This method queries the database and returns any results as in a Cursor object. The query() method we mainly use takes the following parameters:

- [String]: The name of the table to compile the query against
- [String Array]: List of specific column names to return (use null for all)
- [String] The WHERE clause: Use null for all; might include selection args as ?s
- [String Array]: Any selection argument values to substitute in for the ?s in the earlier parameter
- [String] GROUP BY clause: null for no grouping
- [String] HAVING clause: null unless GROUP BY clause requires one
- [String] ORDER BY clause: If null, default ordering used
- [String] LIMIT clause: If null, no limit

Previously, we called the query() method with only one parameter set to the table name, as shown in the following code:

```java
cursor c = mDatabase.query("tbl_books",null,null,null,null,null,null);
```

This is equivalent to the SQL query

```
SELECT * FROM tbl_books;
```

Tip

The individual parameters for the clauses (WHERE, GROUP BY, HAVING, ORDER BY, LIMIT) are all Strings, but you do not need to include the keyword, such as WHERE. Instead, you include the part of the clause after the keyword.

Add a WHERE clause to your query, so you can retrieve one record at a time:

```java
cursor c = mDatabase.query("tbl_books",null,
   "id=?", new String[] {"9"}, null, null, null);
```

This is equivalent to the SQL query

```
SELECT * FROM tbl_books WHERE id=9;
```

Selecting all results might be fine for tiny databases, but it is not terribly efficient. You should always tailor your SQL queries to return only the results you require with no extraneous information included. Use the powerful language of SQL to do the heavy lifting for you whenever possible, instead of programmatically processing results yourself. For example, if you need only the titles of each book in the book table, you might use the following call to the query() method:

```java
String asColumnsToReturn[] = { "title", "id" };
String strSortOrder = "title ASC";
```
Cursor c = mDatabase.query("tbl_books", asColumnsToReturn, null, null, null, null, strSortOrder);

This is equivalent to the SQL query
SELECT title, id FROM tbl_books ORDER BY title ASC;

Executing More Complex Queries Using SQLiteQueryBuilder
As your queries get more complex and involve multiple tables, you should leverage the SQLiteQueryBuilder convenience class, which can build complex queries (such as joins) programmatically.

When more than one table is involved, you need to make sure you refer to columns in a table by their fully qualified names. For example, the title column in the tbl_books table is tbl_books.title. Here we use a SQLiteQueryBuilder to build and execute a simple INNER JOIN between two tables to get a list of books with their authors:

```java
import android.database.sqlite.SQLiteQueryBuilder;
...
SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
queryBuilder.setTables("tbl_books, tbl_authors");
queryBuilder.appendWhere("tbl_books.authorid=tbl_authors.id");
String asColumnsToReturn[] = {
  "tbl_books.title",
  "tbl_books.id",
  "tbl_authors.firstname",
  "tbl_authors.lastname",
  "tbl_books.authorid" 
};
String strSortOrder = "title ASC";

Cursor c = queryBuilder.query(mDatabase, asColumnsToReturn, null, null, null, null, strSortOrder);

First, we instantiate a new SQLiteQueryBuilder object. Then we can set the tables involved as part of our JOIN and the WHERE clause that determines how the JOIN occurs. Then, we call the query() method of the SQLiteQueryBuilder that is similar to the query() method we have been using, except we supply the SQLiteDatabase instance instead of the table name. The earlier query built by the SQLiteQueryBuilder is equivalent to the SQL query:

SELECT tbl_books.title,
   tbl_books.id,
   tbl_authors.firstname,
   tbl_authors.lastname,
   tbl_books.authorid
FROM tbl_books
INNER JOIN tbl_authors on tbl_books.authorid=tbl_authors.id
ORDER BY title ASC;
Executing Raw Queries Without Builders and Column-Mapping

All these helpful Android query utilities can sometimes make building and performing a nonstandard or complex query too verbose. In this case, you might want to consider the `rawQuery()` method. The `rawQuery()` method simply takes a SQL statement string (with optional selection arguments if you include `?`) and returns a `Cursor` of results. If you know your SQL and you don’t want to bother learning the ins and outs of all the different SQL query building utilities, this is the method for you.

For example, let’s say we have a `UNION` query. These types of queries are feasible with the `QueryBuilder`, but their implementation is cumbersome when you start using column aliases and the like.

Let’s say we want to execute the following SQL `UNION` query, which returns a list of all book titles and authors whose names contain the substring `ow` (that is Hallows, Rowling), as in the following:

```
SELECT title AS Name, 'tbl_books' AS OriginalTable
FROM tbl_books
WHERE Name LIKE '%ow%'
UNION
SELECT (firstname||' '|| lastname) AS Name, 'tbl_authors' AS OriginalTable
FROM tbl_authors
WHERE Name LIKE '%ow%'
ORDER BY Name ASC;
```

We can easily execute this by making a string that looks much like the original query and executing the `rawQuery()` method, as shown in the following code:

```java
String sqlUnionExample = "SELECT title AS Name, 'tbl_books' AS OriginalTable
FROM tbl_books WHERE Name LIKE ? UNION SELECT (firstname||' '|| lastname) AS Name, 'tbl_authors' AS OriginalTable
FROM tbl_authors WHERE Name LIKE ? ORDER BY Name ASC;"

Cursor c = mDatabase.rawQuery(sqlUnionExample,
   new String[]{ "%ow%", "%ow%"});
```

We make the substrings (`ow`) into selection arguments, so we can use this same code to look for other substrings' searches.

Closing and Deleting a SQLite Database

Although you should always close a database when you are not using it, you might on occasion also want to modify and delete tables and delete your database.
Deleting Tables and Other SQLite Objects
You delete tables and other SQLite objects in exactly the same way you create them. Format the appropriate SQLite statements and execute them. For example, to drop our tables and triggers, we can execute three SQL statements:

```java
mDatabase.execSQL("DROP TABLE tbl_books;");
mDatabase.execSQL("DROP TABLE tbl_authors;");
mDatabase.execSQL("DROP TRIGGER IF EXISTS fk_insert_book;");
```

Closing a SQLite Database
You should close your database when you are not using it. You can close the database using the `close()` method of your `SQLiteDatabase` instance, like this:

```java
mDatabase.close();
```

Deleting a SQLite Database Instance Using the Application Context
The simplest way to delete a `SQLiteDatabase` is to use the `deleteDatabase()` method of your application `Context`. You delete databases by name and the deletion is permanent. You lose all data and schema information.

```java
deleteDatabase("my_sqlite_database.db");
```

Designing Persistent Databases
Generally speaking, an application creates a database and uses it for the rest of the application's lifetime—by which we mean until the application is uninstalled from the device. So far, we've talked about the basics of creating a database, using it, and then deleting it.

In reality, most mobile applications do not create a database on-the-fly, use them, and then delete them. Instead, they create a database the first time they need it and then use it. The Android SDK provides a helper class called `SQLiteOpenHelper` to help you manage your application's database.

To create a SQLite database for your Android application using the `SQLiteOpenHelper`, you need to extend that class and then instantiate an instance of it as a member variable for use in your application. To illustrate how to do this, let's create a new Android project called PetTracker.

Tip
Many of the code examples provided in this section are taken from the PetTracker application. The source code for the PetTracker application is provided for download on the book's websites.
Keeping Track of Database Field Names

You’ve probably realized by now that it is time to start organizing your database fields programmatically to avoid typos and such in your SQL queries. One easy way you do this is to make a class to encapsulate your database schema in a class, such as PetDatabase, shown here:

```java
import android.provider.BaseColumns;

public final class PetDatabase {
    private PetDatabase() {}

    public static final class Pets implements BaseColumns {
        private Pets() {}
        public static final String PETS_TABLE_NAME = "table_pets";
        public static final String PET_NAME = "pet_name";
        public static final String PET_TYPE_ID = "pet_type_id";
        public static final String DEFAULT_SORT_ORDER = "pet_name ASC";
    }

    public static final class PetType implements BaseColumns {
        private PetType() {}
        public static final String PETTYPE_TABLE_NAME = "table_pettypes";
        public static final String PET_TYPE_NAME = "pet_type";
        public static final String DEFAULT_SORT_ORDER = "pet_type ASC";
    }
}
```

By implementing the BaseColumns interface, we begin to set up the underpinnings for using database-friendly user interface controls in the future, which often require a specially named column called _id to function properly. We rely on this column as our primary key.

Extending the SQLiteOpenHelper Class

To extend the SQLiteOpenHelper class, we must implement several important methods, which help manage the database versioning. The methods to override are onCreate() and onUpgrade() and optionally onDowngrade() and onOpen(). We use our newly defined PetDatabase class to generate appropriate SQL statements, as shown here:

```java
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteOpenHelper;

```
class PetTrackerDatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "pet_tracker.db";
    private static final int DATABASE_VERSION = 1;

    PetTrackerDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " +PetType.PETTYPE_TABLE_NAME+ " (" +
                PetType._ID + " INTEGER PRIMARY KEY AUTOINCREMENT ," +
                PetType.PET_TYPE_NAME + " TEXT" +
            ");
        db.execSQL("CREATE TABLE " + Pets.PETS_TABLE_NAME + " (" +
                Pets._ID + " INTEGER PRIMARY KEY AUTOINCREMENT ," +
                Pets.PET_NAME + " TEXT," +
                Pets.PET_TYPE_ID + " INTEGER" // FK to pet type table +
            ");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Housekeeping here.
        // Implement how to "move" your application data
        // during an upgrade of schema versions.
        // Move or delete data as required. Your call.
    }

    @Override
    public void onOpen(SQLiteDatabase db) {
        super.onOpen(db);
    }
}

Now we can create a member variable for our database like this:

PetTrackerDatabaseHelper mDatabase = new
    PetTrackerDatabaseHelper(this_APPLICATION_CONTEXT);

Now, whenever our application needs to interact with its database, we request a valid
database object. We can request a read-only database or a database that we can also write
to. We can also close the database. For example, here we get a database we can write
data to:

SQLiteDatabase db = mDatabase.getWritableDatabase();


**Binding Data to the Application User Interface**

In many cases with application databases, you want to couple your user interface with the data in your database. You might want to fill drop-down lists with values from a database table, or fill out form values, or display only certain results. There are various ways to bind database data to your user interface. You, as the developer, can decide whether to use built-in data-binding functionality provided with certain user interface controls, or build your own user interfaces from the ground up.

**Working with Database Data Like Any Other Data**

If you peruse the PetTracker application provided on the book’s websites, you notice that its functionality includes no magical data-binding features, yet the application clearly uses the database as part of the user interface.

Specifically, the database is leveraged:

- When you fill out the Pet Type field, the AutoComplete feature is seeded with pet types already listed in the `table_pettypes` table (Figure 3.2, left).
- When you save new records using the Pet Entry Form (Figure 3.2, middle).
- When you display the Pet List screen, you query for all pets and use a Cursor to programmatically build a TableLayout on-the-fly (Figure 3.2, right).

![Figure 3.2](image)

Figure 3.2  The PetTracker application: Entry Screen (left, middle) and Pet Listing Screen (right).

This might work for small amounts of data; however, there are various drawbacks to this method. For example, all the work is done on the main thread, so the more records
you add, the slower your application response time becomes. Second, there’s quite a bit of custom code involved to map the database results to the individual user interface components. If you decide you want to use a different control to display your data, you have quite a lot of rework to do. Third, we constantly requery the database for fresh results, and we might be requerying far more than necessary.

Note
Yes, we really named our pet bunnies after data structures and computer terminology. We are that geeky. Null, for example, is a rambunctious little black bunny. Shane enjoys pointing at him and calling himself a Null pointer.

Binding Data to Controls Using Data Adapters
Ideally, you’d like to bind your data to user interface controls and let them take care of the data display. For example, we can use a fancy ListView to display the pets instead of building a TableLayout from scratch. We can spin through our Cursor and generate ListView child items manually, or even better, we can simply create a data adapter to map the Cursor results to each TextView child within the ListView.

The PetTracker2 application behaves much like the PetTracker sample application, except that it uses the SimpleCursorAdapter with ListView and an ArrayAdapter to handle AutoCompleteTextView features.

Tip
The source code for subsequent upgrades to the series of PetTracker applications is provided for download on the book’s websites.

Binding Data Using SimpleCursorAdapter
Let’s now look at how we can create a data adapter to mimic our Pet Listing screen, with each pet’s name and species listed. We also want to continue to have the ability to delete records from the list.

A ListView container can contain children such as TextView objects. In this case, we want to display each pet’s name and type. We therefore create a layout file called pet_item.xml that becomes our ListView item template:

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

Storing Structured Data Using SQLite Databases
Next, in our main layout file for the Pet List, we place our ListView in the appropriate place on the overall screen. The ListView portion of the layout file might look something like this:

```xml
<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/petList" android:divider="#000" />
```

Now to programmatically fill our ListView, we must take the following steps:
1. Perform our query and return a valid Cursor (a member variable).
2. Create a data adapter that maps the Cursor columns to the appropriate TextView controls within our pet_item.xml layout template.
3. Attach the adapter to the ListView.

In the following code, we perform these steps:

```java
SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
queryBuilder.setTables(Pets.PETS_TABLE_NAME +", " +
        PetType.PETTYPE_TABLE_NAME);
queryBuilder.appendWhere(Pets.PETS_TABLE_NAME + "." +
        Pets.PET_TYPE_ID + "=" + PetType.PETTYPE_TABLE_NAME + "." +
        PetType._ID);
String asColumnsToReturn[] = { Pets.PETS_TABLE_NAME + "." +
        Pets.PET_NAME, Pets.PETS_TABLE_NAME + "." + Pets._ID, PetType.PETTYPE_TABLE_NAME + "." +
        PetType.PET_TYPE_NAME });
mCursor = queryBuilder.query(mDB, asColumnsToReturn, null, null,
    null, null, Pets.DEFAULT_SORT_ORDER);
startManagingCursor(mCursor);
ListAdapter adapter = new SimpleCursorAdapter(this,
    R.layout.pet_item, mCursor,
    new String[] { "pet_type", "pet_name", "pet_id" },
    new int[] { R.id.TextView_PetType, R.id.TextView_PetName, R.id.TextView_PetId } );
```

Chapter 3  Leveraging SQLite Application Databases
new String[] {Pets.PET_NAME, PetType.PET_TYPE_NAME},
new int[] {R.id.TextView_PetName, R.id.TextView_PetType });

ListView av = (ListView)findViewById(R.id.petList);
av.setAdapter(adapter);

Notice that the _id column and the expected name and type columns appear in the query. This is required for the adapter and ListView to work properly.

Using a ListView (Figure 3.3, left) instead of a custom user interface enables us to take advantage of the ListView control’s built-in features, such as scrolling when the list becomes longer, and the ability to provide context menus as needed. The _id column is used as the unique identifier for each ListView child node. If we choose a specific item on the list, we can act on it using this identifier, for example, to delete the item.

Now we reimplement the Delete functionality by listening for onItemClick() events and providing a Delete Confirmation dialog (Figure 3.3, right):

av.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        final long deletePetId = id;
    }
});
RelativeLayout item = (RelativeLayout) view;
TextView nameView = (TextView) item
    .findViewById(R.id.TextView_PetName);
String name = nameView.getText().toString();
new AlertDialog.Builder(PetTrackerListActivity.this)
    .setMessage("Delete Pet Record for " + name + "?")
    .setPositiveButton("Delete",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,
                int which) {
                deletePet(deletePetId);
                mCursor.requery();
            }
        }).show();
}
);

Note that within the PetTracker2 sample application, we also use an ArrayAdapter
to bind the data in the pet_types table to the AutoCompleteTextView on the Pet
Entry screen. Although our next example shows you how to do this in a preferred man-
ner, we left this code in the PetTracker sample to show you that you can always intercept
the data your Cursor provides and do what you want with it. In this case, we create a
String array for the AutoText options by hand. We use a built-in Android layout
resource called android.R.layout.simple_dropdown_item_1line to specify what
each individual item within the AutoText listing looks like. You can find the built-in
layout resources provided within your appropriate Android SDK version’s resource
subdirectory.

A Note on Design
In this example, we’ve followed the traditional Android use of context menus with the
press-and-hold, as the SDK provides for. However, in Android 4.0, the design guidelines
have changed to recommend using press-and-hold for selection. See
http://developer.android.com/design/patterns/new-4-0.html (http://goo.gl/aBH6n) for
more information. When you are using contextual menus, be sure the dialog contains the
context of what item the action is being taken on, such as the name as we’ve shown here.

Storing Nonprimitive Types (Such as Images) in the Database
Because SQLite is a single file, it makes little sense to try to store binary data in the data-
base. Instead store the location of data, as a file path or a URI in the database, and access
it appropriately.
Summary
There are a variety of different ways to store and manage application data on the Android platform. The method you use depends on what kind of data you need to store. Application-specific SQLite databases are secure and efficient mechanisms for structured data storage. You now know how to design persistent data-access mechanisms in your Android application, and you also learned how to bind data from various sources to user interface controls, such as ListView objects.

References and More Information
Android Dev Guide: “Data Storage”:
  http://d.android.com/guide/topics/data/data-storage.html
Android SDK Documentation for the android.database.sqlite package:
SQLite website:
  http://www.sqlite.org/index.html
SQLzoo.net:
  http://sqlzoo.net/
This page intentionally left blank
Symbols

3D objects, drawing, 325
- coloring vertices, 326
- complex objects, 327-328
- lighting scenes, 329-330
- texturing objects, 331-332
- vertices, 325-326

A

accessibility features, 150-151
accessing
- images on devices, 66-67
- Internet, 170
  - handling network operations asynchronously, 174-179
  - HttpURLConnection, 171-172
  - parsing XML from the network, 172-174
  - reading data from the web, 170-171
  - retrieving Android network status, 179-180
  - secondary logs, LogCat, 455
accessories, USB, 268
Accessory Development Kit (ADK), 149-150
Account Manager, 405-406
  - synchronizing data with sync adapters, 406-407
accounts
  - creating Google accounts for analytics, 429-430
  - managing with Account Manager, 405-406
acquiring references to sensors, 256
acting as content type handlers, 381-382
action bars, 105-106
  advanced features, 114
  application icon clicks, 112-113
  contextual action mode, 114
  creating basic action bars, 106-109
  customizing, 110-111
  screens that do not require action bars, 114
action tag, 384
ActionScript 3, 196
Activity class, 108
  implementing to process intents, 383
activity tag, 384
ad and market referrals, tracking, 438
ADB (Android Debug Bridge), 447
  backup services, 455-456
  bug reports, 456
  copying files, 450
  directing commands to specific devices, 448
  installing applications, 451
  installing custom binaries via the shell, 459-461
  listing connected devices and emulators, 447
  LogCat, 452
    accessing secondary logs, 455
    displaying log information, 452
    filtering log information, 453-454
    including date and time with log data, 452
    redirecting log output to files, 454
    monkey tool, 456-459
  reinstalling applications, 451
  retrieving files from devices or emulators, 450
  sending files to devices or emulators, 450
  shell commands, issuing, 449-450
  starting ADB server process, 448
  stopping ADB server process, 448
  uninstalling applications, 452
using shell to inspect SQLite databases, 456
using shell to stress test applications, 456-458
ADB commands, listing, 448
ADB server process, 448
ADB shell, sqlite3, 464
addGlobalFocusChangeListener() method, 129
addGlobalLayoutListener() method, 129
addHelper() method, 409
adding
  browser chrome with WebChromeClient, 188-189
  features to WebView, 186
  libraries to Eclipse projects, 431
addJavascriptInterface() method, 194
addOnTouchModeChangeListener() method, 27
ADK (Accessory Development Kit), 149-150
advanced action bar features, action bars, 114
AIDL files, 25
AIR apps, building for Android, 196
altering data in tables, SQLite, 473
alternative resources
  internationalization, 416-417
    changing language settings, 418-420
    leveraging, 161
alternatives to C2DM, 403
Amazon Appstore in-app billing APIs, leveraging, 426
Amazon Kindle Fire, developing applications for, 166
analytics, creating Google accounts for, 429-430
Android, extending web application functionality to, 190-195
Android Application server side, integrating C2DM services, 402-403
Android applications, developing for Google TV, 165-166
Android Backup Service, registering with, 408

Android Beam, 269
  configuring manifest files, 272
  enabling sending, 270-271
  receiving messages, 271-272

Android client side, integrating C2DM services, 402

Android Debug Bridge. See ADB

Android Design, 99-100

Android Lint Tool, 151

Android manifest file, configuring for sensors, 255

Android manifest files, configuring
  for App Widgets, 373-374
  for live wallpaper, 379
  for searches, 394-395

Android Market campaign tracking, 438

Android Market in-app billing APIs, leveraging, 425-426

Android NDK (Native Development Kit), 355
  creating projects, 357-358
    calling native code from Java, 358-359
    exceptions with native code, 360-362
    native activities, 362
    parameters and return values, 359-360
  exploring, 357
  improving graphics performance, 362-363
  installing, 356
  running sample applications, 357
  versus RenderScript, 363
  when to use, 355-356

Android network status, retrieving, 179-180

Android SDK, OpenGL ES APIs, 317

Android Software Development Kit (SDK), 17

android.speech.RecognizerIntent, 152

animation, 301
  frame-by-frame animation, 302-303
  interpolators, 313-314

property animation, 309
  as XML resources, 310-311
  defining programmatically, 311
  starting programmatically, 312
tweened animation, 304-305
  loading, 306
  moving transformations, 308
  programmatically, 305
  rotating transformations, 307
  scaling transformations, 308
  transparency transformations, 307
  as XML resources, 304

ANR (Application Not Responding) events, 9

anti-aliasing, Paint, 281

anti-piracy tips, 445-446

API Level 11, execute() method, 12

APIs
  debug API key, 207-208
  OpenGL ES, 317

App Widgets, 366-367
  becoming hosts, 375
  configuring Android manifest files, 373-374
  creating, 367
    configurations, 367-368
    determining whether configuration activities are required, 369
    providers, 369
    remote views, 370-371
    update services, 372-373
    updating, 372
    installing, 374-375

application broadcasts, securing, 80

application context, SQLite database instances
  creating, 36
  deleting, 47

application database files, finding on device file systems, 36

application icon clicks, action bars, 112-113

application lifecycle, cursors, 42
Application manifest files, registering backup agents, 412

Application Not Responding events, 9

Application threads
OpenGl threads talking to application threads, 333-334
talking to OpenGl threads, 335-337

Applications
ADB
installing, 451
reinstalling, 451
uninstalling, 452
collecting data from, Google Analytics SDK, 432
configuring for OpenGl ES 2.0, 341
developing for Amazon Kindle Fire, 166
developing for Google TV, 164-165
developing for tables, 162-163
enabling searches, 386-387
enhancing, 365-366
internationalizing, 415
alternative resources, 416-417
changing language settings, 418-420
implementing locale support programmatically, 421
publishing for foreign users, 422
AppWidgetProvider class, 369
implementing, 369-370

Archived data, wiping (ADB), 456
Arcs, drawing, 294-295
Asynchronous processing, importance of, 9-10
AsyncTask class, 10-12, 174-175
Attracting users, device diversity, 161
Audio, 233
playing, 235
recording, 233-234
sharing, 236
AudioManager, 235
AUTOINCREMENT, creating tables, 470

Backup agents
implementing, 409
backup helpers for files, 410-412
backup helpers for shared preferences, 409
providing backup agent implementation, 409
registering in application manifest files, 412

Backup helpers for files, implementing, 410-412

Backup helpers for shared preferences, implementing, 409

Backup operations, forcing ADB, 455

Backup services, 407
ADB, 455
forcing backup operations, 455
forcing restore operations, 456
wiping archived data, 456
backup and restoring application data, 412
requesting backups, 412
requesting restores, 413
implementing backup agents, 409
backup helpers for files, 410-412
backup helpers for shared preferences, 409
providing backup agent implementation, 409
registering backup agents in application manifest files, 412
remote backup services, 408
registering with Android Backup Service, 408

BackupManager, 412
backups, requesting, 412
basic broadcasts, sending, 74
basic searches, enabling, 387
batteries, monitoring, 258-260
BatteryManager object, 260
BATTERY_STATS permission, 259
binding data
to application user interfaces, 50-51
with data adapters, 51
with simpleCursorAdapter, 51-54

bitmaps, 287
drawing on Canvas, 287
scaling graphics, 287
transforming with matrices, 287-288

blinking lights, notifications, 92-93

blocking operations, 9

Bluetooth, 263-264
checking for existence of, 264
discovering devices, 265
enabling, 264
establishing connections, 266
querying for paired devices, 265
BluetoothAdapter class, 264
BluetoothDevice class, 264
BluetoothServerSocket class, 264
BluetoothSocket class, 264

BroadcastReceiver object, 31

broadcasts
ordered broadcasts, 73
receiving, 75
handling incoming broadcasts, 77-79
registering to receive, 76
registering to receive dynamically, 76-77
registering to receive statically, 77
necuring application broadcasts, 80
sending, 73-74

Browser application, 183

browser chrome, adding with
WebChromeClient, 188-189

browsing
web with WebView, 183-184
WebKit APIs, 190

bug reports, ADB, 456

capture() method, 224

C

C2DM (Cloud to Device Messaging), 399
alternatives to, 403
integrating
on Android Application server side,
402-403
on Android client side, 402
limitations of, 400-401
message flow, 400
overview, 399
sample applications, 403
signing up for, 401

calculated columns, SQLite, 474-476
subqueries, 476

calibrating sensors, 258

call state, requesting, 240-242

callbacks, 22

CallVoidMethod(), 361

Camera class, 225

cameras, 220
assigning images as wallpaper, 227-228
capturing still images, 220-225
choosing from device cameras, 228-229
face detection, 233
mode settings, 225
parameters, 225
sharing images, 226-227
video, 229
playing, 231
recording, 229-230
zooming, 226

CameraSurfaceView, 221-224
cancelDiscovery() method, 265

Canvas, 279-281
drawing bitmaps, 287
linear gradients, 282
radical gradients, 283
sweep gradients, 283

capture() method, 224
capturing still images with cameras, 220-225
choosing from device cameras, 228-229
Chrome to Phone app, 403
circles, drawing, 293-294
classes
  Activity, 108
  AppWidgetProvider, 369
    implementing, 369-370
  AsyncTask, 10-12, 174-175
  BluetoothAdapter, 264
  BluetoothDevice, 264
  BluetoothServerSocket, 264
  BluetoothSocket, 264
  Camera, 225
  CameraSurfaceView, 221, 224
  ContextMenu, 103
  Cursor, 41
  CursorAdapter, 69
  GameAreaView, 133-136
  GeomagneticField, 258
  GestureDetector, 132-133
  GestureListener, 140
  GLSurfaceView.Renderer class, 317
  GPSatellite, 216
  IntentService, 29-30, 32
  ItemizedOverlay, 215
  JavaScriptExtensions, 192
  Loader, 14
  Matrix, 287
  Parcelable, 26-29
  PrefListenerService, 372
  RSSurfaceView, 350
  ScrubActivity, 112
  SensorManager, 255
  Shape, 289
  ShapeDrawable, 289
  SimpleDataUpdateService, 372

SQLiteDatabase
deleting records, 39
inserting records, 38
updating records, 39
SQLiteOpenHelper, 48-49
Thread, 13-14, 175-176
UriMatcher, 60-61
UsbManager, 268
ViewTreeObserver, 128
WebChromeClient, 192
cleaning up OpenGL ES, 337
clearing
  logs (LogCat), 454
  notifications, 90
clicks, listening for long clicks, 129-130
clipboard, 126
copying data to system clipboard, 127
pasting data to system clipboard, 127
close() method, 47
closing SQLite databases, 47
Cloud to Device Messaging. See C2DM
collecting data from applications, Google Analytics SDK, 432
color, Paint, 281
coloring vertices, 326
commands, ADB commands
directing to specific devices, 448
listing, 448
compatibility
  notifications, 84
  OpenGL, 316-317
complex objects, drawing, 327-328
composite primary keys, SQLite, 472
configurations, App Widgets, 367-368
configuring
  Android manifest file for sensors, 255
  Android manifest files
    for App Widgets, 373-374
    for live wallpaper, 379
    for searches, 394-395
applications for OpenGL ES 2.0, 341
ProGuard for your apps, 443-444
SQLite database properties, 37
connected devices and emulators, listing in ADB, 447
connecting to SQLite databases, sqlite3, 464
connections, establishing between devices (Bluetooth), 266
content
loading into WebView, 184-186
locating with URIs, 67
content providers, 57
accessing images on devices, 66-67
data columns, 59
data URIs, 59
data-binding to gallery control, 69-70
enhancing applications, 65
implementing interfaces, 58
implementing methods, 59
delete() method, 63-64
getType() method, 64-65
insert() method, 61-62
query() method, 59-60
update() method, 62-63
locating content using URIs, 67
retrieving data with CursorLoader, 68-69
retrieving gallery images and saving to databases, 71
updating manifest files, 63
UriMatcher class, 60-61
content type handlers, acting as, 381-382
context menus, 103-105
Context.getSystemService() method, 269
Context.startService() method, 18
ContextMenu class, 103
contextual action mode, 114
controlling
hardware acceleration, 298
services, 23-24
copying
data to system clipboard, 127
files, ADB, 450
createBitmap() method, 287
createScript() method, 350
Cursor class, 41
Cursor objects, 42
CursorAdapter class, extending, 69
CursorLoader, retrieving content provider data, 68-69
cursors, SQLite databases, 41-42
custom binaries, installing via shell (ADB), 459-461
custom view control, rendering to, 350-352
customizing
action bars, 110-111
notifications, 94-96

dalvik
Dalvik Debug Monitor Service (DDMS), 241
data, 50
binding
to application user interfaces, 50-51
with data adapters, 51
with SimpleCursorAdapter, 51-54
collecting from applications, Google Analytics SDK, 432
copying to system clipboard, 127
extracting in SQLite databases, 42-43
pasting to system clipboard, 127
reading from the web, 170-171
sensor data, reading, 256-257
synchronizing with sync adapters, 406-407
data adapters, binding data, 51
data binding to gallery control, 69-70
data columns, defining, 59
data connection speed, monitoring, 243
data URIs, defining, 59
designing flexible user interfaces, 160
screen space, 161-162
device file system, finding application
database files, 36
device orientation, determining, 258
device sensors, 254-255
acquiring references to, 256
calibrating, 258
configuring Android manifest file for
sensors, 255
determining orientation, 258
finding true north, 258
reading sensor data, 256-257
devices
ADB, 450
discovering, 265
dictionaries, user dictionaries, 126
directing ADB commands to specific
devices, 448
discovering devices, Bluetooth, 265
dispatch() method, 432
displaying
images from network resources, 177-179
log information, LogCat, 452
divideMessages() method, 248
doAlert() function, 193
doBackup() method, 412
doConsoleLog() function, 193
doInBackground() method, 11
doRestore() methods, 412
doServiceStart() method, 21
doSetFromText() function, 193
doToast() function, 193
double-tap gesture, 133
drag and drop framework, gestures, 143
Drawable, 215
drawing
3D objects, 325
coloring vertices, 326
complex objects, 327-328
lighting scenes, 329-330
texturing objects, 331-332
vertices, 325-326
shapes, 291
arcs, 294-295
ovals and circles, 293-294
paths, 295-297
rectangles and squares, 291
rectangles with rounded corners, 291-293
drawing on the screen, 279
Canvas, 279-281
linear gradients, 282
OpenGL ES, 323-324
Paint, 279-281, 284
radical gradients, 283
sweep gradients, 283

E
easy OpenGL ES, GLSurfaceView, 337-340
Eclipse, AIDL files, 25
Eclipse projects, adding libraries to, 431
eCommerce, Google Analytics SDK, 436
logging events, 436-437
reviewing reports, 437
EGL, initializing, 321-323
EGLConfig object, 340
eglDestroyContext(), 337
eglDestroySurface(), 337
eglMakeCurrent(), 337
emulators
ADB, 450
locating, 200
enhancing applications, 365-366
error reports, ProGuard, 444
events
eCommerce events, logging, 436-437
handling WebView events with WebViewClient, 186-187
logging Google Analytics SDK, 432
user events. See user events

ExceptionOccurred(), 361
exceptions with native code, Android NDK, 360-362
execSQL() method, 37
execute() method, AsyncTask class, 11
executing
commands on command line, SQLite, 468
queries, 44
raw queries, 46
SQLiteQueryBuilder, 45
exporting databases, SQLite, 466-468
extending web application functionality to Android, 190-195
extracting data, SQLite databases, 42-43

F
face detection, 233
features, adding to WebView, 186
field names, databases, 48
files
copying in ADB, 450
retrieving from devices or emulators, 450
sending to devices or emulators, 450
filtering log information, LogCat, 453-454
finding application database files on device file system, 36
finding true north, sensors, 258
fine-tuning hardware acceleration, 298
Flash, 195
building AIR apps for Android, 196
enabling applications, 195
fling gesture, 133
focus changes, listening for, 130-131
fonts, default fonts, 284-285
foreign keys, SQLite, 472
foreign users, publishing applications for, 422
formatNumber() method, 243
frame-by-frame animation, 302-303
Galaxy Nexus Android device, 255

gallery control, data binding to, 69-70
game design, 162
GameAreaView class, 133-136
Geocoder object, 201-203
genocoding locations, 200-204
GeomagneticField class, 258
GestureDetector class, 132-133
GestureListener class, 140
GestureOverlayView, 132
gestures, 131
detecting user motions within a view, 131-132
drag and drop framework, 143
making them natural, 142
multi-touch gestures, 139-141
single-touch gestures, 132-138
getAccessoryList() method, 268
getAddressLine() method, 202
getAvailableLocales() method, 421
getBestProvider() method, 199
getCenter() method, 216
getDuration(), 232
getFeatureName() method, 202
getLocality() method, 202
getRoaming() method, 242
getSystemService() method, 240
getTextBounds method, 287
getAddress() method, content providers, 64-65
GL (Graphics Library), 315
initializing, 323
glCompileShader(), 344
GLDebugHelper, 323
glDrawArrays(), 325
glDrawElements(), 325
global positioning services. See GPS

global search
enabling, 395
updating search configurations, 396
updating search settings, 397
glShaderSource() method, 344
GLSurfaceView, 317
GLSurfaceView (easy OpenGL ES), 337-340
GLSurfaceView.Render class, 317
glUseProgram() method, 344
Google Analytics, creating Google accounts for, 429-430
Google Analytics Dashboard, 433-435
Google Analytics SDK
adding the library to your Eclipse project, 431
collecting data from applications, 432
eCommerce, 436
logging events, 436-437
reviewing reports, 437
gathering statistics, 438-439
logging different events, 432
protecting users’ privacy, 439
tracking ad and market referrals, 438
Google APIs, 201
Google TV
developing Android applications for, 165-166
developing applications for, 164-165
optimizing web applications for, 165
GPS (global positioning services), 197-198
determining location of device, 198-200
locating your emulator, 200
GpsSatellite class, 216
GPXPoint class, 27
gradients, Paint, 281
graphics
bitmap graphics, scaling, 287
improving performance (Android NDK), 362-363
guidelines, user interfaces, 99
**hardware**
- Android Beam, 269
  - configuring manifest files, 272
  - enabling sending, 270-271
  - receiving messages, 271-272
- batteries, 258-260
- Bluetooth, 263-264
  - checking for existence of, 264
  - discovering devices, 265
  - enabling, 264
  - establishing connections, 266
  - querying for paired devices, 265
  - device sensors, 254-255
    - acquiring reference to, 256
    - calibrating, 258
    - configuring Android manifest file, 255
    - determining device orientation, 258
    - finding true north, 258
    - reading sensor data, 256-257
- interacting with, 253-254
- USB, 267
  - accessories, 268
  - hosts, 269
- Wi-Fi, 273
  - monitoring, 274-276
  - Wi-Fi Direct, 273

**hardware acceleration**, 297
- controlling, 298
- fine-tuning, 298

**hosts**
- App Widgets, 375
- USB, 269

**HTTP**, 170
- HttpURLConnection, 171-172
  - reading data from the web, 170-171

**HttpURLConnection**, 171-172

**icons, text notifications**, 85-86

**images**
- accessing on devices, 66-67
- assigning as wallpaper, 227-228
- displaying from network resources, 177-179
- retrieving, 71
- saving to databases, 71
- sharing, 226-227

**IME (Input Method Editors)**, 123

**implementing**
- backup agents, 409
  - backup helpers for files, 410-412
  - backup helpers for shared preferences, 409
  - providing backup agent implementation, 409
  - registering backup agents in application manifest files, 412
- content provider interfaces, 58
- content provider methods, 59
  - delete() method, 63-64
  - getType() method, 64-65
  - insert() method, 61-62
  - query() method, 59-60
  - update() method, 62-63

**importing databases, SQLite**, 466-468

**improving graphics performance, Android NDK**, 362-363

**in-app billing**, 424-425, 445
- defined, 423-424
- leveraging
  - Amazon Appstore in-app billing APIs, 426
  - Android Market in-app billing APIs, 425-426
  - in-app billing APIs, 427
  - PayPal in-app billing APIs, 426

**incoming broadcasts, handling**, 77-79

**init()**, 347, 350
initializing

EGL, OpenGL ES, 321-323
GL, OpenGL ES, 323

Input Method Editors (IME), 123

insert() method, content providers, 61-62

inserting
data into tables, SQLite, 471
records, SQLiteDatabase class, 38

installing
Android NDK, 356
App Widgets, 374-375
applications, ADB, 451
custom binaries via shell, ADB, 459-461
live wallpaper, 379

instances
SQLite database instances, creating with application context, 36
SQLite databases, deleting with application context, 47

Integrated Raster Imaging System Graphics Library (IRIS GL), 315

integrating C2DM services
on Android application server side, 402-403
on Android client side, 402

intent actions, 382

Intent extras, 31
intent filter, registering, 384
IntentFilter, 32

intents
implementing Activity to process, 383
mapping, 205

IntentService class, 29-32
interacting with hardware, 253-254

interfaces
content provider interfaces, implementing, 58
designing flexible user interfaces, 160

internationalizing applications, 415
alternative resources, 416-420
implementing locale support programmatically, 421

Internet, accessing, 170
handling network operations asynchronously, 174-179
HttpURLConnection, 171-172
parsing XML from the network, 172-174
reading data from the web, 170-171
retrieving Android network status, 179-180

interpolators, 313-314

invalidate() method, 140

IRemoteInterface, 25-26
IRIS GL (Integrated Raster Imaging System Graphics Library), 315

isDiscovering() method, 265
issuing shell commands, 449-450

ItemizedOverlay class, 215
iterating rows of query results, SQLite databases, 42-43

J

Java, calling native code (Android NDK), 358-359
JavaScriptExtensions class, 192
JNIenvironment object, 360
JOIN, querying multiple tables (SQLite), 474

K

Khronos Group, 315
Kindle Fire, 161

L

languages, 416-417
changing settings, 418-420

layout, designing with WebView, 184

LBS (location-based services), 197, 216
geocoding locations, 200-204
GPS (global positioning services), 197-198
determining location of device, 198-200
locating your emulator, 200
mapping locations, 204
  debug API key, 207-208
  intents, 205
  marking the spot, 211-216
  panning map view, 209
  views, 206-207
  zooming map view, 210
leveraging
  Amazon Appstore in-app billing APIs, 426
  Android Market in-app billing APIs, 425-426
  billing APIs, 427
  hardware acceleration, 297
  License Verification Library, 444-445
  OpenGL ES in Android, 316
  PayPal in-app billing APIs, 426
  style inheritance, 117-119
libraries adding to Eclipse projects, 431
License Verification Library (LVL), leveraging, 444-445
lighting scenes, 329-330
lights, blinking (notifications), 92-93
limitations
  of C2DM, 400-401
  of SQLite, 469
linear gradients, 282
listening
  for events on the entire screen, 128-129
  for focus changes, 130-131
  for long clicks, 129-130
  for touch mode changes, 127-128
listing
  ADB commands, 448
  connected devices and emulators, ADB, 447
ListView container, 51
live wallpaper, 375
  configuring Android manifest files, 379
  creating, 376
  creating configurations, 378
  creating service, 376-377
  installing, 379
  service engine, implementing, 377-378
live wallpaper service, implementing, 377
live wallpaper service engine, implementing, 377-378
loadAndCompileShader() method, 344
Loader class, 14
loading
  animations, 306
  content into WebView, 184-186
locale support, implementing programmatically, 421
locating content using URLs, 67
location-based services. See LBS
LocationListener, 199
LocationListener object, 22-23
LocationManager object, 199
locations
  geocoding, 200-204
  mapping, 204
    debug API key, 207-208
    intents, 205
    marking the spot, 211-216
    panning map view, 209
    views, 206-207
    zooming map view, 210
log information, filtering (ADB), 453-454
LogCat, 144
ADB, 452
  accessing secondary logs, 455
  displaying log information, 452
  filtering log information, 453-454
  including date and time with log data, 452
  redirecting log output to files, 454
logging
  eCommerce events, 436-437
  events, Google Analytics SDK, 432
logs, clearing (LogCat), 454
long clicks, listening for, 129-130
LVL (License Verification Library), 444-445

managing
accounts with Account Manager, 405-406
WebView state, 189

manifest files
configuring for Android Beam, 272
updating, 65

map view
panning, 209
zooming, 210
MapActivity, 206
MapController object, 207
mapping locations, 204
debug API key, 207-208
intents, 205
marking the spot, 211-216
panning map view, 209
views, 206-207
zooming map view, 210
MapView object, 206
MapView XML, 206
marking the spot, mapping locations, 211-216

Matrix class, 287
matrices, transforming bitmaps, 287-288
measureText() method, 287
menus, 100
context menus, 103-105
options menus, 100-103
popup menus, 105
message flow, C2DM, 400
MIME types, 381-382
mobile networking, 169
Mobile Payment Libraries, 426
mode settings, cameras, 225
modifying WebView settings with WebSettings, 186
monitoring
batteries, 258-260
signal strength and data connection speed, 243
Wi-Fi, 274-276
monkey tool, ADB, 456-459
moving transformations, animations, 308
multi-touch gestures, 139-141
multimedia, 219-220
audio, 233
playing, 235
recording, 233-234
sharing, 236

cameras, 220
assigning images as wallpaper, 227-228
capturing still images, 220-225
choosing from device cameras, 228-229
face detection, 233
mode settings, 225
parameters, 225
sharing images, 226-227
video. See video
zooming, 226
ringtones, 238
searching, 236-237

native activities, Android NDK, 362
native code, exceptions (Android NDK), 360-362
Native Development Kit. See Android NDK
Near Field Communications (NFC), 269
network operations, handling asynchronously, 174-176
displaying images from network resources, 177-179
network resources, displaying images from, 177-179
networking, strict mode, 170
NFC (Near Field Communications), 269
noises, notifications, 93
notification queue, 86-87
NotificationManager, 85
notifications, 83-84
   blinking lights, 92-93
clearing, 90
compatibility, 84
customizing, 94-96
designing, 96
noises, 93
status bar, 84-85
   NotificationManager service, 85
test notification with icons, 85-86
updating, 88-89
vibrating phones, 91-92
notify() method, 86
notifyChange() method, 64
notifying users, 83-84

obfuscating with ProGuard, 443
   error reports, 444
objects
   SQLite databases, deleting, 47
texturing, 331-332
onActivityResult() method, 154
onAnimateMove() method, 138
onBackup() method, 411
onClick() method, 233
onCreate() method, 215
onCreateContextMenu() method, 104
onCreateLoader() method, 14
onDestroy() method, 22
onDetachedFromWindow() method, 351
onDoubleTap, 132
onDoubleTapEvent, 132
onDown, 132
onDraw() method, 139, 211
onFling, 133
onInit() method, 156
onKeyDown() method, 336
onKeyUp() method, 336
onLoaderReset() method, 14
onLoadFinished() method, 14
onLocationChanged() method, 200
onLongPress, 132
onOptionsItemSelected() method, 102, 112
onPause() method, 189
onPostExecute() method, 11
onPreExecute() method, 11
onProgressUpdate() method, 11
onReceive() callback method, 32
onReceive() method, 369
onRestore() method, 411
onScale() helper method, 140
onScroll, 133
onSensorChanged() method, 257
onServiceConnected() method, 26
onServiceDisconnected() method, 26
onServiceStateChanged method, 241
onShowPress, 132
onSingleTapConfirmed, 132
onSingleTapUp, 132
onStart() method, 18
onStartCommand() method, 18
onSurfaceCreate(), 342
onSurfaceCreated() method, 340
onTouchEvent() method, 139
onTrackballEvent(), 143
onUpdate() method, 372
OpenGL, 315
   device compatibility, 316-317
   enabling application threads to talk to, 335-337
   enabling OpenGL threads to talk to application threads, 333-334
OpenGL ES

OpenGL ES, 315

- APIs in Android SDK, 317
- cleaning up, 337
- drawing 3D objects
  - coloring vertices, 326
  - complex objects, 327-328
  - lighting scenes, 329-330
  - texturing objects, 331-332
  - vertices, 325-326
- handling tasks manually, 318
- drawing on the screen, 323-324
- initializing EGL, 321-323
- initializing GL, 323
- starting threads, 319-321
- SurfaceView, 318-319
- leveraging in Android, 316

OpenGL ES 1.x, 341

OpenGL ES 2.0, 341

- configuring applications for, 341
- requesting surfaces, 341-345
- openOrCreateDatabase() method, 36

optimizing web applications for Google TV, 165

options menus, 100-103

ordered broadcasts, 73

- sending, 74

OrientationEventListener, 145

ovals, drawing, 293-294

P

Paint, 279-281, 284

Paint gradients, 281

paired devices, querying for, 265

panning map view, 209

parameters
  - Android NDK, 359-360
  - cameras, 225

Parcelable class, 26-29

- parsing XML from the network, 172-174
- pasting data to system clipboard, 127
- paths, drawing, 295-297
- pauseTimers(), 189

PayPal in-app billing APIs, leveraging, 426

performance of graphics, improving (Android NDK), 362-363

permission
  - to access phone state information, 240
  - gaining to send and receive SMS messages, 244

persistent database, designing, 47

- field names, 48
  - SQLiteOpenHelper class, 48-49

PetTracker, 47

PetTracker3, 67

phone calls
  - making, 248-250
  - receiving, 251

phone numbers, 243-244

phone state information, gaining permission to access, 240

phones, vibrating notifications, 91-92

piracy. See software piracy

playing
  - audio, 235
  - video, 231

popup menus, 105

PrefListenerService class, 372

privacy, Google Analytics, 439

programmatically defining
  - property animations, 311
  - shape drawables, 290
  - tweened animations, 305

programmatically implementing locale support, 421

ProGuard, 442-443

- configuring for your apps, 443-444
- error reports, 444
projects, Android NDK, 357-358
  calling native code from Java, 358-359
  exceptions with native code, 360-362
  native activities, 362
  parameters and return values, 359-360
property animation, 309
  defining programmatically, 311
  starting programmatically, 312
  as XML resources, 310-311
providing backup agent
  implementation, 409
publishing applications for foreign
  users, 422
publishProgress() method, 11
reading
  data from the web, 170-171
  sensor data, 256-257
receiving
  Android Beam messages, 271-272
  broadcasts, 75
    handling incoming broadcasts, 77-79
    registering to receive, 76
    registering to receive dynamically, 76-77
    registering to receive statically, 77
  phone calls, 251
  SMS, 247-248
RecognizerIntent, 153-154
recording
  audio, 233-234
  video, 229-230
records
  deleting in SQLiteDatabase class, 39
  inserting in SQLiteDatabase class, 38
  updating in SQLiteDatabase class, 39
rectangles, drawing, 291
  with rounded corners, 291-293
redirecting log output to files, LogCat, 454
references to sensors, acquiring, 256
registerForContextMenu(), 103
registering
  with Android Backup Service, 408
  backup agents in application manifest
    files, 412
  intent filter, 384
  to receive broadcasts, 76
    dynamically, 76-77
    statically, 77
registerListener() method, 258
reinstalling applications, ADB, 451
release() method, 235
remote backup services, 408
remote interfaces, implementing
  24-26
remote view, creating App Widgets, 370-371
RemoteViews object, 373
rendering to custom view control, 350-352
RenderScript, 345-346
  versus Android NDK, 363
defining functionality, 346-350
  rendering to custom view control, 350-352
RenderScriptGL object, 349
reports
eCommerce reports, reviewing, 437
error reports, ProGuard, 444
requesting
backups, 412
call state, 240-242
OpenGL ES 2.0 surfaces, 341-345
restores, 413
searches, 390-391
service information, 242
restore operations, forcing (ADB), 456
restores, requesting, 413
retrieving
Android network status, 179-180
content provider data with CursorLoader, 68-69
files to devices or emulators, 450
gallery images, 71
return values, Android NDK, 359-360
reviewing eCommerce reports, 437
ringtones, 238
root(), 347-348
rotating transformations, animations, 307
rsgClearColor(), 348
RSSurfaceView class, 350
ScanResult object, 275
scenes, lighting, 329-330
schema objects, creating SQLite, 37-38
screen orientation changes, 144-146
screen requirements, text, 287
screen space, device diversity, 161-162
screens that do not require action bars, 114
scroll gesture, 133
ScrubActivity class, 112
SDK (Software Development Kit), 17
search activity, creating, 392-394
search configurations, creating, 387
search suggestions, enabling, 387-390
searches
  Android manifest files, configuring, 394-395
  basic searches, enabling, 387
  creating configurations, 387
  enabling in applications, 386-387
  global search
    enabling, 395
    updating search configurations, 396
    updating search setting, 397
  making application content searchable,
    385-386
  requesting, 390-391
  suggestions, enabling, 387-390
  voice search, enabling, 390
searching multimedia, 236-237
secondary logs, accessing (LogCat), 455
secure coding practices, software piracy, 442
securing application broadcasts, 80
SELECT, querying tables (SQLite), 471
sending
  Android Beam, 270-271
  broadcasts, 73-74
  files to devices or emulators, 450
  SMS, 245-246
sendMultipartTestMessage() method, 248
sensor data, reading, 256-257
saving images to databases, 71
ScaleGestureDetector, 132, 140
scaling bitmap graphics, 287
scaling transformations, animations, 308
SensorManager class, 255
sensors. See device sensors
sequential tweened animations, 305
service information, requesting, 242
service lifecycle, 18
<service> tags, 23
services
controlling, 23-24
creating, 18-23
determining when to use, 17
lifecycles of, 18
Session Initiated Protocol (SIP), 251
setAccuracy() method, 199
setAnchorView() method, 231
setARGB() method, 281
setColor() method, 281
displayHomeAsUpEnabled() method, 113
setImageUri() method, 70
setInitialScale() method, 186
setInterpolator() method, 314
setMediaController() method, 231
setNotificationUri() method, 60
setOnFocusChangeListener() method, 130
setOnNdefPushCompleteCallback() method, 271
setShader() method, 282
setTables() method, 60
setTheme(), 119
SGI (Silicon Graphics), 315
Shape class, 289
shape drawables, defining
programmatically, 290
as XML resources, 289
ShapeDrawable class, 289
shapes, 289
defining shape drawables
as XML resources, 289
programmatically, 290
drawing, 291
arcs, 294-295
ovals and circles, 293-294
paths, 295-297
rectangles and squares, 291
rectangles with rounded corners, 291-293
sharing
audio, 236
images, 226-227
shell, ADB (inspecting SQLite
databases), 456
shell commands, issuing, 449-450
shell sessions, 449
shells, stress testing applications (ADB),
456-458
Short Message Service. See SMS
signal strength, monitoring, 243
signing up for C2DM, 401
Silicon Graphics (SGI), 315
SimpleCursorAdapter, binding data, 51-54
SimpleDataUpdateService class, 372
simultaneous tweened animations, 305
single-touch gestures, 132-138
SIP (Session Initiated Protocol), 251
SMS (Short Message Service), 239, 244
gaining permission to send and receive
messages, 244
receiving, 247-248
sending, 245-246
SmsManager.divideMessage() method, 248
software keyboards, 123, 126
choosing, 124-125
software piracy, 441
anti-piracy tips, 445-446
configuring ProGuard for your apps,
443-444
leveraging License Verification Library,
444-445
obfuscating with ProGuard, 442-443
error reports, 44
secure coding practices, 442
speak() method, 157
speech recognition, 149-154
SQLite, 463
executing commands on command line, 468
importing/exporting databases, 466-468
limitations of, 469
sqlite3 command-line interface, 464
connecting to SQLite databases, 464
exploring databases, 465-466
launching ADB shell, 464
sqlite3 commands, 469
SQLiteOpenHelper class
inserting records, 38
updating records, 39
SQLite database instances, deleting with application context, 47
SQLite databases
closing, 47
configuring properties, 37
creating, 36
instances with application context, 36
tables and schema objects, 37-38
deleting tables and objects, 47
designing, 470
foreign keys and composite primary keys, 472
inserting data into tables, 471
querying multiple tables with JOIN, 474
querying tables with SELECT, 471
subqueries for calculated columns, 476
SQLite database instances, deleting with application context, 47
SQLiteOpenHelper class
inserting records, 38
updating records, 39
SQLite database instances, deleting with application context, 47
SQLiteOpenHelper class
inserting records, 38
updating records, 39
SQLiteDatabase class
deleting records, 39
inserting records, 38
SQLiteOpenHelper class, 48-49
SQLiteQuery class
executing queries, 45
squares, drawing, 291
startActivity() method, 250
startActivityForResult() method, 154
startDiscovery() method, 265
starting
ADB server process, 448
OpenGL ES threads, 319-321
startManagingCursor(), 42
startService() method, 18
startSmoothZoom() method, 226
statistics, gathering with Google Analytics, 438-439
status bar, notifications, 84-85
NotificationManager service, 85
text notifications with icons, 85-86
stdio sprintf() call, 359
still images, capturing with cameras, 220-225
stop() method, 235
stopping ADB server process, 448
stopScan() method, 274
stopSelf() method, 18, 22
startService() method, 24
### tables
- altering and updating data, SQLite, 473
- creating with AUTOINCREMENT (SQLite), 470
- deleting in SQLite, 476
- inserting data, SQLite, 471
- querying with SELECT (SQLite), 471
- querying multiple, with JOIN (SQLite), 474
- SQLite database, creating, 37-38
- SQLite databases, deleting, 47

### tablets, developing applications for, 162-163

### tags, 384

### telephony utilities, 239
- gaining permission to access phone state information, 240
- monitoring signal strength and data connection speed, 243
- phone calls
  - making, 248-250
  - receiving, 251
- phone numbers, 243-244
- requesting call state, 240-242
- requesting service information, 242
- SIP (Session Initiated Protocol), 251
- SMS, 244
  - gaining permission to send and receive messages, 244
  - receiving, 247-248
  - sending, 245-246

### TelephonyManager, 240

### text, 284
- default fonts and typefaces, 284-285
- screen requirements, 287
- typefaces, custom, 285-286

### text notifications with icons, 85-86

### text prediction, 126

### Text-To-Speech services (TTS), 149, 155-157
TextToSpeech.OnInitListener

TextToSpeech.OnInitListener, 156
textual input methods, 123
  clipboard, 126
    copying data to, 127
    pasting data to, 127
  software keyboards, 123, 126
    choosing, 124-125
  text prediction, 126
  user dictionaries, 126
texturing objects, 331-332
themes, 119-121
Thread class, 13-14, 175-176
threads
  application threads, talking to OpenGL threads, 333-337
  enabling OpenGL threads to talk to application threads, 333-334
  OpenGL ES, starting, 319-321
toggleFPSDisplay() method, 336
touch mode, listening for changes, 127-128
trackballs, 143
tracking ad and market referrals, 438
trackTransactions() method, 437
transactions, SQLite databases, 40-41
transformations, tweened animation
  moving transformations, 308
  rotating transformations, 307
  scaling transformations, 308
  transparency transformations, 307
transforming bitmaps with matrixes, 287-288
transparency transformations, animations, 307
ture north, finding, 258
tweened animation, 304-305
  loading, 306
  moving transformations, 308
  programmatically, 305
  rotating transformations, 307
  scaling transformations, 308
typefaces, 284-285
  custom, 285-286
  TYPE_ACCELEROMETER, 255
  TYPE_AMBIENT_TEMPERATURE, 255
  TYPE_LIGHT, 255
  TYPE_MAGNETIC_FIELD, 255
  TYPE_PRESSURE, 255
  TYPE_PROXIMITY, 255
  TYPE_RELATIVE_HUMIDITY, 255
uninstalling applications, ADB, 452
update services, App Widgets, 372-373
update() method, content providers, 62-63
updating
  App Widgets, 372
  data in tables, SQLite, 473
  manifest files, 65
  notifications, 88-89
  records, SQLiteDatabase class, 39
  search configurations for global searches, 396
  search settings for global searches, 397
UriMatcher, 59-61
URIs, locating content on Android systems, 67
USB, 267
  accessories, 268
  hosts, 269
UsbManager class, 268
user dictionaries, 126
user events, 127
  listening for
    events on the entire screen, 128-129
    focus changes, 130-131
    long clicks, 129-130
    touch mode changes, 127-128
user interface guidelines, 99
**user interfaces**

- action bars, 105-106
- advanced features, 114
- application icon clicks, 112-113
- contextual action mode, 114
- creating basic, 106-109
- customizing, 110-111
- screens that do not require action bars, 114
- binding data to, 50-51
- menus, 100
  - context menus, 103-105
  - options menus, 100-103
  - popup menus, 105
- styles, 114
  - creating, 115-117
  - leveraging style inheritance, 117-119
- themes, 119-121

**user motions, detecting within views, 131-132**

**users**

- attracting with device diversity, 161
- notifying, 83-84

**vertices, coloring, 326**

**vibrating phones, notifications, 91-92**

**vibration patterns, 91**

**video, 229**

- playing, 231
- recording, 229-230

**videoRecorder object, 230**

**view animation, 304**

**views**

- detecting user motions, 131-132
- mapping, 206-207

**ViewTreeObserver class, 128**

**voice search, enabling, 390**

**W**

**wallpaper, assigning images, 227-228**

**web**

- browsing with WebView, 183-184
- reading data from, 170-171

**web applications, optimizing for Google TV, 165**

**web extensions, building with WebKit, 190**

**WebChromeClient, adding browser chrome, 188-189**

**WebChromeClient class, 192**

**WebKit**

- APIs, browsing, 190
- building web extensions, 190
- extending functionality to Android, 190-195

**WebSettings, modifying WebView settings, 186**

**WebView**

- adding features to, 186
- browsing the web, 183-184
- designing layouts, 184
- handling events with WebViewClient, 186-187
- loading content into, 184-186
- managing, 189
- modifying settings with WebSettings, 186

**WebViewClient, handling WebView events, 186-187**

**Wi-Fi, 273**

- monitoring, 274-276
- Wi-Fi Direct, 273

**Wi-Fi Direct, 273**

**WifiConfiguration object, 276**

**WifiManager object, 276**

**wiping, archived data (ADB), 456**

**world Magnetic Model, 258**

**writeToParcel() method, 28**
XML, parsing from the network, 172-174
XML resources
  property animations, 310-311
  shape drawables, 289
  tweened animations, 304
XMPP (Extensible Messaging and Presence Protocol), 403

zooming
  cameras, 226
  map view, 210