Shane Conder
Lauren Darcey

Android™ Wireless Application Development

Developer's Library

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@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data:

Conder, Shane, 1975-
Android wireless application development / Shane Conder, Lauren Darcey.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-62709-4 (pbk. : alk. paper) 1. Mobile computing. 2. Android (Electronic resource) 3. Application software--Development. 4. Wireless communication systems. I. Darcey, Lauren, 1977- II. Title.
QA76.59.C65 2009
621.3845'6–dc22

2009027111

ISBN-13: 978-0-321-62709-4
ISBN-10: 0-321-62709-1
Text printed in the United States on recycled paper at

RR Donnelley Crawfordsville, Indiana

First printing August 2010

Editor-in-Chief
Mark Taub

Acquisitions Editor
Trina MacDonald

Development Editor
Songlin Qiu

Managing Editor
Patrick Kanouse

Project Editor
Bethany Wall

Copy Editor
Apostrophe Editing Services

Indexer
Tim Wright

Proofreader
Jovana San Nicolas-Shirley

Technical Reviewers
Dan Galpin
Tony Hillerson
Ronan Schwarz

Publishing Coordinator
Olivia Basegio

Multimedia Developer
Dan Scherf

Designer
Gary Adair

Compositor
Bronkella Publishing, LLC

Introduction

Pioneered by the Open Handset Alliance and Google, Android is a hot, new, free, open source mobile platform making waves in the wireless world. This book provides 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, this book provides everything you need to make a smooth transition from traditional software development to mobile development—specifically, its most promising new platform: Android.

Who Should Read This Book

This book includes tips for successful mobile development based on our years in the mobile industry and covers everything you need 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 can work for you.

This book was written for three primary audiences:

- **Software developers** who want to learn to develop professional Android applications

  The bulk of this book is primarily targeted at software developers with Java experience but not necessarily 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 phone. 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 this book to use for fun and profit.

Key Questions Answered in this Book

This book answers the following questions:

1. What is Android?
2. How is Android different from other mobile technologies and how can developers take advantage of these differences?
3. How do developers use the Eclipse Development Environment for Java to develop and debug Android applications on the emulator and handsets?
4. How are Android applications structured?
5. How do developers design robust user interfaces for mobile—specifically for Android?
6. What capabilities does the Android SDK have and how can developers use them?
7. How does the mobile development process differ from traditional desktop development?
8. What development strategies work best for Android development?
9. What do managers, developers, and testers need to look for when planning, developing, and testing a mobile development application?
10. How do mobile teams design bulletproof Android applications for publication?
11. How do mobile teams package Android applications for deployment?
12. How do mobile teams make money from Android applications?

How This Book Is Structured

This book is divided into seven parts. The first five parts are primarily of interest to developers; Parts VI and VII provide lots of helpful information for project managers and quality assurance personnel as well as developers.

Here is an overview of the various parts in this book:

- Part I: An Overview of Android

  Part I provides an introduction to Android, explaining how it differs from other mobile platforms. You become familiar with the Android SDK and tools, install the

development tools, and write and run your first Android application—on the emulator and on a handset.

- Part II: Android Application Design Essentials

  Part II introduces the design principles necessary to write Android applications. You learn how Android applications are structured and how to include resources like strings, graphics, and user interface components in your projects.

- Part III: Android User Interface Design Essentials

  Part III dives deeper into how user interfaces are designed in Android. You learn about the core user interface element in Android: the `View`. You'll also learn about the basic drawing and animation abilities provided in the Android SDK.

- Part IV: Using Common Android APIs

  Part IV is a series of chapters, each devoted to a deeper understanding of the most important APIs within the Android SDK, such as the data and storage APIs (including file and database usage as well as content providers), networking, telephony, Location-Based Services (LBS), multimedia and 3D graphics APIs, and the optional hardware APIs available.

- Part V: More Android Application Design Principles

  Part V covers more advanced Android application design principles such as notifications and services.

- Part VI: Deploying Your Android Application to the World

  Part VI covers the software development process for mobile, from start to finish, with tips and tricks for project management, software developers, and quality assurance personnel.

- Part VII: Appendixes

  Part VII includes several helpful quick-start guides for the Android development tools: the emulator, ADB and DDMS, and a SQLite tutorial.

Android Development Environment Used in This Book

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

- Windows Vista SP1 and Mac OS X 10.5.6
- Eclipse Java IDE Version 3.4 (Ganymede)
- Eclipse JDT plug-in and Web Tools Platform (WTP)
- Sun Java SE Development Kit (JDK) 6 Update 10
- Android SDK Version 1.1 R1 and Version 1.5 R1

> **Note**
>
> Many of the examples have also been tested using the Android SDK on Fedora 8 and Windows XP installations.

Supplementary Materials Available

The source code that accompanies this book is provided on a CD at the end of this book and on the publisher Web site: www.informit.com/title/9780321627094.

Lauren Darcey and Shane Conder run a book blog at http://androidbook.blogspot.com, where you can find the latest news about the Android topics covered here.

Where to Find More Information

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

- Android Developer Website—The Android SDK and developer reference site
  http://developer.android.com/
- Open Handset Alliance—Android manufacturers, operators, and developers
  www.openhandsetalliance.com/
- Android Market—Buy and sell Android applications
  www.android.com/market/
- anddev.org—An Android developer forum
  www.anddev.org
- Google Team Android Apps—Open source Android applications
  http://apps-for-android.googlecode.com/
- FierceDeveloper—A weekly newsletter for wireless developers
  www.fiercedeveloper.com/
- FierceWireless—A daily digest for the wireless industry
  www.fiercewireless.com/
- FierceMobileContent—A daily digest of mobile content and marketing
  www.fiercemobilecontent.com/
- Wireless Developer Network—Daily news on the wireless industry
  www.wirelessdevnet.com/
- Developer.com—A developer-oriented site with mobile articles
  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.

This book also presents information in the following sidebars:

> **Tip**
>
> Tips provide useful information or hints related to the current text.

> **Note**
>
> Notes provide additional information that might be interesting or relevant.

> **Caution**
>
> Cautions provide hints or tips about pitfalls that might be encountered and how to avoid them.

> **What's New in Android SDK 1.5**
>
> This special sidebar calls out the new and interesting features available in the latest version of the Android SDK.

Contacting the Authors

Both Lauren and Shane welcome your comments. If you have questions or feedback regarding this book, please visit our book blog at http://androidbook.blogspot.com or email us at androidwirelessdev@gmail.com.

1

Introducing Android

The mobile development community is at a tipping point. Mobile users demand more choice, more opportunities to customize their phones, and more functionality. Mobile operators want to provide value-added content to their subscribers in a manageable and lucrative way. Mobile developers want the freedom to develop the powerful mobile applications users demand with minimal roadblocks to success. Finally, handset manufacturers want a stable, secure, and affordable platform to power their devices. Upuntil now single mobile platform has adequately addressed the needs of all the parties.

Enter Android, which is a potential game-changer for the mobile development community. An innovative and open platform, Android is well positioned to address the growing needs of the mobile marketplace.

This chapter explains what Android is, how and why it was developed, and where the platform fits in to the established mobile marketplace.

A Brief History of Mobile Software Development

To understand what makes Android so compelling, we must examine how mobile development has evolved and how Android differs from competing platforms.

Way Back When

Remember way back when a phone was just a phone? When we relied on fixed landlines? When we ran for the phone instead of pulling it out of our pocket? When we lost our friends at a crowded ballgame and waited around for hours hoping to reunite? When we forgot the grocery list (Figure 1.1) and had to find a payphone or drive back home again?

Those days are long gone. Today, commonplace problems like these are easily solved with a one-button speed dial or a simple text message like "WRU?" or "20?" or "Milk and?"

Figure 1.1 Mobile phones have become a crucial shopping accessory.

Our mobile phones keep us safe and connected. Nowadays, we roam around freely, relying on our phones not only to keep in touch with friends, family, and coworkers, but also to tell us where to go, what to do, and how to do it. Even the most domestic of events seem to revolve around my mobile phone.

Consider the following true, but slightly enhanced for effect, story:

Once upon a time, on a warm summer evening, I was happily minding my own business cooking dinner in my new house in rural New Hampshire when a bat swooped over my head, scaring me to death.

The first thing I did—while ducking—was pull out my cell and send a text message to my husband, who was across the country at the time: "There's a bat in the house!"

My husband did not immediately respond (a divorce-worthy incident, I thought at the time), so I called my Dad and asked him for suggestions on how to get rid of the bat.

He just laughed.

Annoyed, I snapped a picture of the bat with my phone and sent it to my husband and my blog, simultaneously guilt-tripping him and informing the world of my treacherous domestic wildlife encounter.

Finally, I Googled "get rid of a bat" and followed the helpful do-it-yourself instructions provided on the Web for people in my situation. I also learned that late August is when baby bats often leave the roost for the first time and learn to fly. Newly aware that I had a baby bat on my hands, I calmly got a broom and managed to herd the bat out of the house.

Problem solved—and I did it all with the help of my trusty cell phone, the old LG VX9800.

My point here? Mobile phones can solve just about *anything*—and we rely on them for *everything* these days.

You notice that I used half a dozen different mobile applications over the course of this story. Each application was developed by a different company and had a different user interface. Some were well designed; others not so much. I paid for some of the applications, and others came on my phone.

As a user, I found the experience functional, but not terribly inspiring. As a mobile developer, I wished for an opportunity to create a more seamless and powerful application that could handle all I'd done and more. I wanted to build a better bat trap, if you will.

Before Android, mobile developers faced many roadblocks when it came to writing applications. Building the better application, the unique application, the competing application, the hybrid application, and incorporating many common tasks such as messaging and calling in a familiar way were often unrealistic goals.

To understand why, let's take a brief look at the history of mobile software development.

"The Brick"

The Motorola DynaTAC 8000X was the first commercially available cell phone. First marketed in 1983, it was 13 x 1.75 x 3.5 inches in dimension, weighed about 2.5 pounds, and allowed you to talk for a little more than half an hour. It retailed for $3,995, plus hefty monthly service fees and per-minute charges.

We called it "The Brick," and the nickname stuck for many of those early mobile phones we alternatively loved and hated. About the size of a brick, with a battery power just long enough for half a conversation, these early mobile handsets were mostly seen in the hands of traveling business execs, security personnel, and the wealthy. First-generation mobile phones were just too expensive. The service charges alone would bankrupt the average person, especially when roaming.

Early mobile phones were not particularly full featured. (Although, even the Motorola DynaTAC, shown in Figure 1.2, had many of the buttons we've come to know well, such as the SEND, END, and CLR buttons.) These early phones did little more than make and receive calls and, if you were lucky, there was a simple contacts application that wasn't impossible to use.

These first-generation mobile phones were designed and developed by the handset manufacturers. Competition was fierce and trade secrets were closely guarded. Manufacturers didn't want to expose the internal workings of their handsets, so they usually developed the phone software in-house. As a developer, if you weren't part of this inner circle, you had no opportunity to write applications for the phones.

It was during this period that we saw the first "time-waster" games begin to appear. Nokia was famous for putting the 1970s video game *Snake* on some of its earliest monochrome phones. Other manufacturers followed, adding games like Pong, Tetris, and Tic-Tac-Toe.

Figure 1.2 The first commercially available mobile phone: the Motorala DynaTAC.

These early phones were flawed, but they did something important—they changed the way people thought about communication. As mobile phone prices dropped, batteries improved, and reception areas grew, more and more people began carrying these handy devices. Soon mobile phones were more than just a novelty.

Customers began pushing for more features and more games. But, there was a problem. The handset manufacturers didn't have the motivation or the resources to build every application users wanted. They needed some way to provide a portal for entertainment and information services without allowing direct access to the handset.

And what better way to provide these services than the Internet?

Wireless Application Protocol (WAP)

It turned out allowing direct phone access to the Internet didn't scale well for mobile.

By this time, professional Web sites were full color and chock full of text, images, and other sorts of media. These sites relied on JavaScript, Flash, and other technologies to enhance the user experience and were often designed with a target resolution of 800x600 pixels and higher.

When the first clamshell phone, the Motorola StarTAC, was released in 1996, it merely had a LCD 10-digit segmented display. (Later models would add a dot-matrix type

display.) Meanwhile, Nokia released one of the first slider phones, the 8110—fondly referred to as "The Matrix Phone," as the phone was heavily used in films. The 8110 could display four lines of text with 13 characters per line. Figure 1.3 shows some of the common phone form factors.

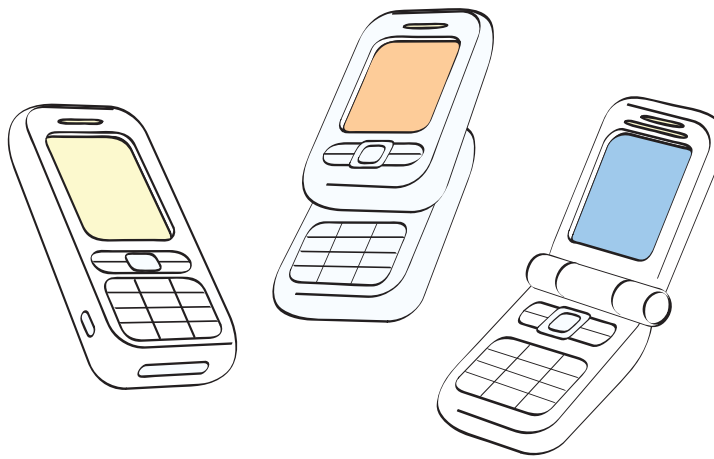

Figure 1.3 Various mobile phone form factors: the candy bar, the slider, and the clamshell.

With their postage stamp-sized low-resolution screens and limited storage and processing power, these phones couldn't handle the data-intensive operations required by traditional Web browsers. The bandwidth requirements for data transmission were also costly to the user.

The Wireless Application Protocol (WAP) standard emerged to address these concerns. Simply put, WAP was a stripped-down version of HTTP, which is the backbone protocol of the Internet. Unlike traditional Web browsers, WAP browsers were designed to run within the memory and bandwidth constraints of the phone. Third-party WAP sites served up pages written in a markup language called Wireless Markup Language (WML). These pages were then displayed on the phone's WAP browser. Users navigated as they would on the Web, but the pages were much simpler in design.

The WAP solution was great for handset manufacturers. The pressure was off—they could write one WAP browser to ship with the handset and rely on developers to come up with the content users wanted.

The WAP solution was great for mobile operators. They could provide a custom WAP portal, directing their subscribers to the content they wanted to provide, and rake in the data charges associated with browsing, which were often high.

Developers and content providers didn't deliver. For the first time, developers had a chance to develop content for phone users, and some did so, with limited success.

Most of the early WAP sites were extensions of popular branded Web sites, such as CNN.com and ESPN.com, looking for new ways to extend their readership. Suddenly phone users accessed the news, stock market quotes, and sports scores on their phones.

Commercializing WAP applications was difficult, and there was no built-in billing mechanism. Some of the most popular commercial WAP applications that emerged during this time were simple wallpaper and ringtone catalogues, allowing users to personalize their phones for the first time. For example, the users browsed a WAP site and requested a specific item. They filled out a simple order form with their phone number and their handset model. It was up to the content provider to deliver an image or audio file compatible with the given phone. Payment and verification were handled through various premium-priced delivery mechanisms such as Short Message Service (SMS), Enhanced Messaging Service (EMS), Multimedia Messaging Service (MMS), and WAP Push.

WAP browsers, especially in the early days, were slow and frustrating. Typing long URLs with the numeric keypad was onerous. WAP pages were often difficult to navigate. Most WAP sites were written once for all phones and did not account for individual phone specifications. It didn't matter if the end-user's phone had a big color screen or a postage stamp-sized monochrome one; the developer couldn't tailor the user's experience. The result was a mediocre and not very compelling experience for everyone involved.

Content providers often didn't bother with a WAP site and instead just advertised SMS short codes on TV and in magazines. In this case, the user sent a premium SMS message with a request for a specific wallpaper or ringtone, and the content provider sent it back. Mobile operators generally liked these delivery mechanisms because they received a large portion of each messaging fee.

WAP fell short of commercial expectations. In some markets, such as Japan, it flourished, whereas in others, like the United States, it failed to take off. Handset screens were too small for surfing. Reading a sentence fragment at a time, and then waiting seconds for the next segment to download, ruined the user experience, especially because every second of downloading was often charged to the user. Critics began to call WAP "Wait and Pay."

Finally, the mobile operators who provided the WAP portal (the default home page loaded when you started your WAP browser) often restricted which WAP sites were accessible. The portal allowed the operator to restrict the number of sites users could browse and to funnel subscribers to the operator's preferred content providers and exclude competing sites. This kind of walled garden approach further discouraged third-party developers, who already faced difficulties in monetizing applications, from writing applications.

Proprietary Mobile Platforms

It came as no surprise when users wanted more—they will always want more.

Writing robust applications such as graphic-intensive video games with WAP was nearly impossible. The 18-year-old to 25-year-old sweet-spot demographic—the kids

with the disposable income most likely to personalize their phones with wallpapers and ringtones—looked at their portable gaming systems and asked for a device that was both a phone and a gaming device or a phone and a music player. They argued that if devices such as Nintendo's Game Boy could provide hours of entertainment with only five buttons, why not just add phone capabilities? Others looked to their digital cameras, Palms, Blackberries, iPods, and even their laptops and asked the same question. The market seemed to be teetering on the edge of device convergence.

Memory was getting cheaper; batteries were getting better; and PDAs and other embedded devices were beginning to run compact versions of common operating systems such as Linux and Windows. The traditional desktop application developer was suddenly a player in the embedded device market, especially with Smartphone technologies such as Windows Mobile, which they found familiar.

Handset manufacturers realized that if they wanted to continue to sell traditional handsets, they needed to change their protectionist policies pertaining to handset design and expose their internal frameworks, at least, to some extent.

A variety of different proprietary platforms emerged—and developers are still actively creating applications for them. Some Smartphone devices ran Palm OS (now Garnet OS) and RIM Blackberry OS. Sun Microsystems took its popular Java platform and J2ME emerged (now known as Java Micro Edition [Java ME]). Chipset maker Qualcomm developed and licensed its Binary Runtime Environment for Wireless (BREW). Other platforms, such as Symbian OS, were developed by handset manufacturers such as Nokia, Sony Ericsson, Motorola, and Samsung. The Apple iPhone OS (OS X iPhone) joined the ranks in 2008. Figure 1.4 shows several different phones, all of which have different development platforms.

Figure 1.4 Phones from various mobile device platforms.

Many of these platforms have associated developer programs. These programs keep the developer communities small, vetted, and under contractual agreements on what they can and cannot do. These programs are often required and developers must pay for them.

Each platform has benefits and drawbacks. Of course, developers love to debate over which platform is "the best." (Hint: It's usually the platform we're currently developing for.)

The truth is no one platform has emerged victorious. Some platforms are best suited for commercializing games and making millions—if your company has brand backing. Other platforms are more open and suitable for the hobbyist or vertical market applications. No mobile platform is best suited for all possible applications. As a result, the mobile phone has become increasingly fragmented, with all platforms sharing part of the pie.

For manufacturers and mobile operators, handset product lines became complicated fast. Platform market penetration varies greatly by region and user demographic. Instead of choosing just one platform, manufacturers and operators have been forced to sell phones for all the different platforms to compete. We've even seen some handsets supporting multiple platforms. (For instance, Symbian phones often also support J2ME.)

The mobile developer community has become as fragmented as the market. It's nearly impossible to keep track of all the changes in the market. Developer specialty niches have formed. The platform development requirements vary greatly. Mobile software developers work with distinctly different programming environments, different tools, and different programming languages. Porting among the platforms is often costly and not straightforward. Keeping track of handset configurations and testing requirements, signing and certification programs, carrier relationships, and application marketplaces have become complex spin-off businesses of their own.

It's a nightmare for the ACME Company wanting a mobile application. Should they develop a J2ME application? BREW? iPhone? Windows Mobile? Everyone has a different kind of phone. ACME is forced to choose one or, worse, all of the above. Some platforms allow for free applications, whereas others do not. Vertical market application opportunities are limited and expensive.

As a result, many wonderful applications have not reached their desired users, and many other great ideas have not been developed at all.

The Open Handset Alliance

Enter search advertising giant Google. Now a household name, Google has shown an interest in spreading its brand and suite of tools to the wireless marketplace. The company's business model has been amazingly successful on the Internet, and technically speaking, wireless isn't that different.

Google Goes Wireless

The company's initial forays into mobile were beset with all the problems you would expect. The freedoms Internet users enjoyed were not shared by mobile phone subscribers. Internet users can choose from the wide variety of computer brands, operating systems, Internet service providers, and Web browser applications.

Nearly all Google services are free and ad driven. Many applications in the Google Labs suite would directly compete with the applications available on mobile phones. The applications range from simple calendars and calculators to navigation with Google Maps and the latest tailored news from News Alerts—not to mention corporate acquisitions like Blogger and YouTube.

When this approach didn't yield the intended results, Google decided to a different approach—to revamp the entire system upon which wireless application development was based, hoping to provide a more open environment for users and developers: the Internet model. The Internet model allowes users to choose between freeware, shareware, and paid software. This enables free market competition among services.

Forming of the Open Handset Alliance

With its user-centric, democratic design philosophies, Google has led a movement to turn the existing closely guarded wireless market into one where phone users can move between carriers easily and have unfettered access to applications and services. With its vast resources, Google has taken a broad approach, examining the wireless infrastructure from the FCC wireless spectrum policies to the handset manufacturers' requirements, application developer needs, and mobile operator desires.

Next, Google joined with other like-minded members in the wireless community and posed the following question: What would it take to build a better mobile phone?

The Open Handset Alliance (OHA) (Figure 1.5) was formed in November 2007 to answer that very question. The OHA is a business alliance comprised of many of the largest and most successful mobile companies on the planet. Its members include chip makers, handset manufacturers, software developers, and service providers. The entire mobile supply chain is well represented.

Figure 1.5 The Open Handset Alliance.

Working together, OHA members began developing a nonproprietary open standard platform that would aim to alleviate the aforementioned problems hindering the mobile community. They called it the Android project.

Google's involvement in the Android project has been extensive. The company hosts the open source project and provides online documentation, tools, forums, and the

Software Development Kit (SDK). Google has also hosted a number of events at conferences and the Android Developer Challenge, a contest to encourage developers to write killer Android applications—for $10 million dollars in prizes.

Manufacturers: Designing the Android Handsets

More than half the members of the OHA are handset manufacturers, such as Samsung, Motorola, HTC, and LG, and semiconductor companies, such as Intel, Texas Instruments, NVIDIA, and Qualcomm. These companies are helping design the first generation of Android handsets.

The first shipping Android handset—the T-Mobile G1—was developed by handset manufacturer HTC with service provided by T-Mobile. It was released in October 2008. Many other Android handsets are slated for 2009 and early 2010.

Content Providers: Developing Android Applications

When users have Android handsets, they need those killer apps, right?

Google has led the pack, developing Android applications, many of which, like the email client and Web browser, are core features of the platform. OHA members, such as eBay, are also working on Android application integration with their online auctions.

The first Android Developer Challenge received 1,788 submissions—all newly developed Android games, productivity helpers, and a slew of Location-Based Services (LBS). We also saw humanitarian, social networking, and mash-up apps. Many of these applications have debuted with users through the Android Market—Google's software distribution mechanism for Android.

Mobile Operators: Delivering the Android Experience

After you have the phones, you have to get them out to the users. Mobile operators from Asia, North America, Europe, and Latin America have joined the OHA, ensuring a market for the Android movement. With almost half a billion subscribers, telephony giant China Mobile is a founding member of the alliance. Other operators have signed on as well.

Taking Advantage of All Android Has to Offer

Android's open platform has been embraced by much of the mobile development community—extending far beyond the members of the OHA.

As Android phones and applications become more readily available, many in the tech community anticipate other mobile operators and handset manufacturers will jump on

the chance to sell Android phones to their subscribers, especially given the cost benefits compared to proprietary platforms. Already, North American operators, such as Verizon Wireless and AT&T, have shown an interest in Android, and T-Mobile already provides handsets.

If the open standard of the Android platform results in reduced operator costs in licensing and royalties, we could see a migration to open handsets from proprietary platforms such as BREW, Windows Mobile, and even the Apple iPhone. Android is well suited to fill this demand.

Android Platform Differences

Android is hailed as "the first complete, open, and free mobile platform."

- **Complete:** The designers took a comprehensive approach when they developed the Android platform. They began with a secure operating system and built a robust software framework on top that allows for rich application development opportunities.
- **Open:** The Android platform is provided through open source licensing. Developers have unprecedented access to the handset features when developing applications.
- **Free:** Android applications are free to develop. There are no licensing or royalty fees to develop on the platform. No required membership fees. No required testing fees. No required signing or certification fees. Android applications can be distributed and commercialized in a variety of ways.

Android: A Next Generation Platform

Although Android has many innovative features not available in existing mobile platforms, its designers also leveraged many tried-and-true approaches proven to work in the wireless world. It's true that many of these features appear in existing proprietary platforms, but Android combines them in a free and open fashion, while simultaneously addressing many of the flaws on these competing platforms.

The Android mascot is a little green robot, shown in Figure 1.6. You'll see this little guy (girl?) often used to depict Android-related materials.

Android is the first in a new generation of mobile platforms, giving its platform developers a distinct edge on the competition. Android's designers examined the benefits and drawbacks of existing platforms and then incorporate their most successful features. At the same time, Android's designers avoided the mistakes others suffered in the past.

Figure 1.6 The Android mascot.

Free and Open Source

Android is an open source platform. Neither developers nor handset manufacturers pay royalties or license fees to develop for the platform.

The underlying operating system of Android is licensed under GNU General Public License Version 2 (GPLv2), a strong "copyleft" license where any third-party improvements must continue to fall under the open source licensing agreement terms. The Android framework is distributed under the Apache Software License (ASL/Apache2), which allows for the distribution of both open and closed source derivations of the source code. Commercial developers (handset manufacturers especially) can choose to enhance the platform without having to provide their improvements to the open source community. Instead, developers can profit from enhancements such as handset-specific improvements and redistribute their work under whatever licensing they want.

Android application developers have the ability to distribute their applications under whatever licensing scheme they prefer. Developers can write open source freeware or traditional licensed applications for profit and everything in between.

Familiar and Inexpensive Development Tools

Unlike some proprietary platforms that require developer registration fees, vetting, and expensive compilers, there are no upfront costs to developing Android applications.

Freely Available Software Development Kit

The Android SDK and tools are freely available. Developers can download the Android SDK from the Android Web site after agreeing to the terms of the Android Software Development Kit License Agreement.

Familiar Language, Familiar Development Environments

Developers have several choices when it comes to integrated development environments (IDEs). Many developers choose the popular and freely available Eclipse IDE to design and develop Android applications. Eclipse is the most popular IDE for Android development and there is an Android plug-in available for facilitating Android development. Android applications can be developed on the following operating systems:

- Windows XP or Vista
- Mac OS X 10.4.8 or later (x86 only)
- Linux (tested on Linux Ubuntu 6.06 LTS, Dapper Drake)

Reasonable Learning Curve for Developers

Android applications are written in a well-respected programming language: Java.

The Android application framework includes traditional programming constructs, such as threads and processes and specially designed data structures to encapsulate objects commonly used in mobile applications. Developers can rely on familiar class libraries, such as `java.net` and `java.text`. Specialty libraries for tasks like graphics and database management are implemented using well-defined open standards like OpenGL Embedded Systems (OpenGL ES) or SQLite.

Enabling Development of Powerful Applications

In the past, handset manufacturers often established special relationships with trusted third-party software developers (OEM/ODM relationships). This elite group of software developers wrote native applications, such as messaging and Web browsers, which shipped on the handset as part of the phone's core feature set. To design these applications, the manufacturer would grant the developer privileged inside access and knowledge of a handset's internal software framework and firmware.

On the Android platform, there is no distinction between native and third-party applications, enabling healthy competition among application developers. All Android applications use the same libraries. Android applications have unprecedented access to the underlying hardware, allowing developers to write much more powerful applications. Applications can be extended or replaced altogether. For example, Android developers are now free to design email clients tailored to specific email servers such as Microsoft Exchange or Lotus Notes.

Rich, Secure Application Integration

If you recall the bat story I previously shared, you'll note that I accessed a wide variety of phone applications in the course of a few moments: text messaging, phone dialer, camera, email, picture messaging, and the browser. Each was a separate application running on the phone—some built-in and some purchased. Each had its own unique user interface. None were truly integrated.

Not so with Android. One of the Android platform's most compelling and innovative features is well-designed application integration. Android provides all the tools necessary to build a better "bat trap," if you will, by allowing developers to write applications that leverage core functionality such as Web browsing, mapping, contact management, and messaging seamlessly. Applications can also become content providers and share their data among each other in a secure fashion.

Platforms like Symbian have suffered from setbacks due to malware. Android's vigorous application security model helps protect the user and the system from malicious software.

No Costly Obstacles to Publication

Android applications have none of the costly and time-intensive testing and certification programs required by other platforms such as BREW and Symbian.

A "Free Market" for Applications

Android developers are free to choose any kind of revenue model they want. They can develop freeware, shareware, or trial-ware applications, ad-driven, and paid applications. Android was designed to fundamentally change the rules about what kind of wireless applications could be developed. In the past, developers faced many restrictions that had little to do with the application functionality or features:

- Store limitations on the number of competing applications of a given type
- Store limitations on pricing, revenue models, and royalties
- Operator unwillingness to provide applications for smaller demographics

With Android, developers can write and successfully publish any kind of application they want. Developers can tailor applications to small demographics, instead of just large-scale money-making ones often insisted upon by mobile operators. Vertical market applications can be deployed to specific, targeted users.

Because developers have a variety of application distribution mechanisms to choose from, they can pick the methods that work for them instead of being forced to play by others' rules. Android developers can distribute their applications to users in a variety of ways.

- Google developed the Android Market (Figure 1.7), a generic Android application store with a revenue-sharing model.
- Handango.com added Android applications to its existing catalogue using their billing models and revenue sharing model.
- Developers can come up with their own delivery and payment mechanisms.

Mobile operators are still free to develop their own application stores and enforce their own rules, but it will no longer be the only opportunity developers have to distribute their applications.

Figure 1.7 The Android market.

Android might be the next generation in mobile platforms, but the technology is still in its early stages. Early Android developers have had to deal with the typical roadblocks associated with a new platform: frequently revised SDKs, lack of good documentation, and market uncertainties. There are only a handful of Android handsets available to consumers at this time.

On the other hand, developers diving into Android development now benefit from the first-to-market competitive advantages we've seen on other platforms such as BREW and Symbian. Early developers who give feedback are more likely to have an impact on the long-term design of the Android platform and what features will come in the next version of the SDK. Finally, the Android forum community is lively and friendly. Incentive programs, such as the Android Developer Challenge, have encouraged many new developers to dig into the platform.

A New and Growing Platform

What's New in Android 1.5

The much-anticipated Android 1.5 SDK, released in late April 2009, provided a number of substantial improvements to both the underlying software libraries and the Android development tools and build environment. Also, the Android system received some much-needed UI "polish," both in terms of visual appeal and performance.

Although most of these upgrades and improvements were welcome and necessary, the new SDK version did cause some upheaval within the Android developer community. A number of published applications required retesting and resubmission to the Android Marketplace to conform to the new SDK requirements, which were quickly rolled out to all Android phones in the field as a firmware upgrade, rendering older applications obsolete.

The Android Platform

Android is an operating system and a software platform upon which applications are developed. A core set of applications for everyday tasks, such as Web browsing and email, are included on Android handsets.

As a product of the Open Handset Alliance's vision for a robust and open source development environment for wireless, Android is an emerging mobile development platform. The platform was designed for the sole purpose of encouraging a free and open market that all mobile applications phone users might want to have and software developers might want to develop.

Android's Underlying Architecture

The Android platform is designed to be more fault-tolerant than many of its predecessors. The handset runs a Linux operating system, upon which Android applications are executed in a secure fashion. Each Android application runs in its own virtual machine (Figure 1.8). Android applications are managed code; therefore, they are much less likely to cause the phone to crash, leading to fewer instances of device corruption (also called "bricking" the phone, or rendering it useless).

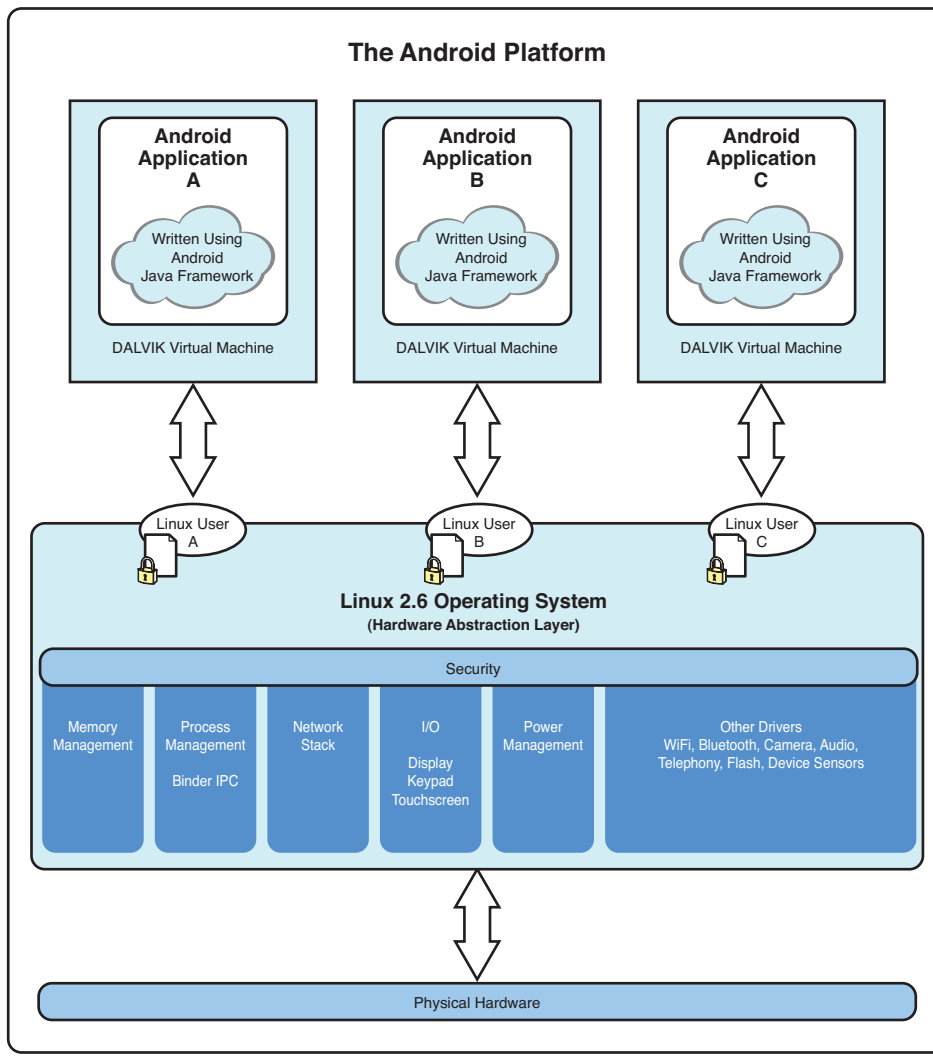

Figure 1.8 Diagram of the Android platform architecture.

The Linux Operating System

The Linux 2.6 kernel (Figure 1.9) handles core system services and acts as a hardware abstraction layer (HAL) between the physical hardware of the handset and the Android software stack.

Figure 1.9 Tux, the Linux kernel mascot.

What's New in Android 1.5

For Android 1.5, the Linux kernel received an upgrade from version 2.6.25 to 2.6.27. Although this type of change might not have an obvious effect for the typical Android developer, it is important to note that the kernel can and will be upgraded frequently. These seemingly minor incremental updates often include major security, performance, and functional features.

Kernel changes often have an impact on the security of the underlying device operating system and provide features and improvements for OEM-level Android device manufacturers. When stable, these features can be exposed to developers as part of an Android SDK upgrade, in the form of new APIs and performance enhancements to existing features.

The Android 1.5 version provides substantial feature enhancements, many of which tie back to features of the upgraded Linux kernel. Although the kernel memory footprint is larger, overall system performance has improved and a number of bugs have been fixed.

Some of the core functions the kernel handles include

- Enforcement of application permissions and security
- Low-level memory management
- Process management and threading
- The network stack
- Display, keypad input, camera, WiFi, Flash memory, audio, and binder (IPC) driver access

Android Application Runtime Environment

Each Android application runs in a separate process, with its own instance of the Dalvik virtual machine (VM). Based on the Java VM, the Dalvik design has been optimized for mobile devices. The Dalvik VM has a small memory footprint and multiple instances of the Dalvik VM can run concurrently on the handset.

Security and Permissions

The integrity of the Android platform is maintained through a variety of security measures.

Applications as Operating System Users

When an application is installed, the operating system creates a new user profile associated with the application. Each application runs as a different user, with its own private files on the file system, a user ID, and a secure operating environment.

The application executes in its own process with its own instance of the Dalvik VM and under its own user ID on the operating system.

Explicitly Defined Application Permissions

To access shared resources on the system, Android applications register for the specific privileges they require. Some of these privileges enable the application to use phone functionality to make calls, access the network, and control the camera and other hardware sensors. Applications also require permission to access shared data containing private and personal information such as user preferences, user's location, and contact information.

Applications might also enforce their own permissions by declaring them for other applications to use. The application can declare any number of different permission types, such as read-only or read-write permissions, for finer control over the application.

Limited Ad-Hoc Permissions

Applications that act as content providers might want to provide some on-the-fly permissions to other applications for specific information they want to share openly. This is done using ad-hoc granting and revoking of access to specific resources using Uniform Resource Identifiers (URIs).

URIs index specific data assets on the system, such as images and text. Here is an example of a URI that provides the phone numbers of all contacts:

```
content://contacts/phones
```

To understand how this permission process works, let's look at an example.

Let's say we've got an application that keeps track of the user's public and private birthday wish lists. If this application wanted to share its data with other applications, it could grant URI permissions for the public wish list, allowing another application permission to access this list without explicitly having to ask for it.

Application Signing for Trust Relationships

All Android applications packages are signed with a certificate, so users know that the application is authentic. The private key for the certificate is held by the developer. This helps establish a trust relationship between the developer and the user. It also allows the developer to control which applications can grant access to one another on the system. No certificate authority is necessary; self-signed certificates are acceptable.

Developing Android Applications

The Android SDK provides an extensive set of application programming interfaces (APIs) that is both modern and robust. Android handset core system services are exposed and accessible to all applications. When granted the appropriate permissions, Android applications can share data among one another and access shared resources on the system securely.

Android Programming Language Choices

Android applications are written in Java (Figure 1.10). For now, the Java language is the developer's only choice on the Android platform. There has been some speculation that other programming languages, such as C++, might be added in future versions of Android.

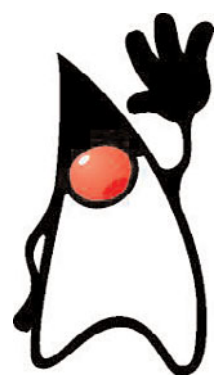

Figure 1.10 Duke, the Java mascot.

No Distinctions Made Between Native and Third-Party Applications

Unlike other mobile development platforms, there is no distinction between native applications and developer-created applications on the Android platform. Provided the application is granted the appropriate permissions, all applications have the same access to core libraries and the underlying hardware interfaces.

Android handsets ship with a set of native applications such as a Web browser and contact manager. Third-party applications might integrate with these core applications and even extend them to provide a rich user experience.

Commonly Used Packages

With Android, mobile developers no longer have to reinvent the wheel. Instead, developers use familiar class libraries exposed through Android's Java packages to perform common tasks such as graphics, database access, network access, secure communications, and utilities (such as XML parsing).

The Android packages include support for

- Common user interface widgets (Buttons, Spin Controls, Text Input)
- User interface layout
- Secure networking and Web browsing features (SSL, WebKit)
- Structured storage and relational databases (SQLite)
- Powerful 2D and 3D graphics (SGL and OpenGL ES 1.0)
- Audio and visual media formats (MPEG4, MP3, Still Images)
- Access to optional hardware such as Location-Based Services (LBS), WiFi, and Bluetooth

Android Application Framework

The Android application framework provides everything necessary to implement your average application. The Android application lifecycle involves the following key components:

- Activities are functions the application performs.
- Groups of views define the application's layout.
- Intents inform the system about an application's plans.
- Services allow for background processing without user interaction.
- Notifications alert the user when something interesting happens.

Android Applications can interact with the operating system and underlying hardware using a collection of managers. Each manager is responsible for keeping the state of some underlying system service. For example, there is a `LocationManager` that facilitates interaction with the location-based services available on the handset. The `ViewManager` and `WindowManager` manage user interface fundamentals.

Applications can interact with one another by using or acting as a `ContentProvider`. Built-in applications such as the Contact manager are content providers, allowing third-party applications to access contact data and use it in an infinite number of ways. The sky is the limit.

Summary

Mobile software development has evolved over time. Android has emerged as a new mobile development platform, building on past successes and avoiding past failures of other platforms. Android was designed to empower the developer to write innovative applications. The platform is open source, with no up-front fees, and developers enjoy many benefits over other competing platforms.

Now it's time to dive deeper and start writing Android code, so you can evaluate what Android can do for you. In the next chapter, we configure the Android development environment and take a brief walk through the Android SDK.

References and More Information

Android Development http://developer.android.com
Open Handset Alliance: http://www.openhandsetalliance.com

Index

Numerics

3D applications, OpenGL ES, 366
drawing, 372-382
EGL, initializing, 370-372
GL, initializing, 372
SurfaceView, creating, 367-368
thread, starting, 369-370

A

aapt (Android Asset Packaging Tool), 96
AbsoluteLayout view class, 193
attributes, 194
example, 194-195
accessing
images programmatically, 112-113
the Internet
Android network status, retrieving, 312-314
HttpURLConnection object, 305
images, displaying from network resource, 310-312
reading web data, 304
threads, using for network calls, 308-310
XML, parsing from network, 305-307
layouts programmatically, 122
resources, 99
secondary logs, 531
styles programmatically, 126
accessing
acquiring, target handsets, 435-436
Activities, 73
activities
belonging to another application, launching, 75-76

Intents, 74–75
broadcasts, 77
creating, 75
filters, configuring, 87
information, passing, 76
launching, 74–75
lifecycle, 77–81
organizing with menus, 76
registering, 86–87
Services, 77
transitions, 81–82
ad revenue, leveraging, 486
ad-hoc application permissions, 26
AdapterView widget
data, binding to, 208
selection events, handling, 209
AdapterView widgets, 206
ArrayAdapter, 206–207
CursorAdapter, 207–208
ListActivity, 209
ADB (Android Debug Bridge), 45, 525
applications, installing/uninstalling, 528
commands, directing to specific devices, 526
commands, listing, 537
connected devices, listing, 525
files, retrieving from devices, 527
LogCat
filtering by event severity, 530
filtering by tag, 530
log information, displaying, 529
log output, redirecting to a file, 531
log, clearing, 530
logging modes, changing, 529
server processes, starting/stopping, 526
adding
location-based services support to applications, 66–68
logging support to Android applications, 63–64
media support to applications, 64–66
records to Content Providers, 289
Snake project to Eclipse workspace, 50–51
addOnTouchModeChangeListener() method, 171
addView() method, 188
ADT, installing, 35-37
alpha transformations, 244
altering table data, SQLite Student grade database, 549
AnalogClock widget, 165
AndAppStore, 485
Android 1.5 SDK, 23
Android application framework, 28, 42
Android Developers Guide, 41
Android Emulator, 43-44
applications,
debugging, 61–63
running, 58–59
instances, calling between, 505–506
location information, configuring, 504
Snake application, running, 54
startup options, 502
configuring from command line, 503
configuring from Eclipse, 503
Android Hierarchy Viewer, 46
Android Market
preparing applications for, 478–479
publishing applications on, 483
refund policy, 483
removing applications from, 483
uploading applications to, 482–483
Android network status, retrieving, 312-314
Android Plug-In for Eclipse, installing, 35, 37
Android SDK
documentation, 41
installing, 34–35
license agreement, 39–41
reporting problems with, 38
sample applications, 47
tabbed interface, 210–212
upgrading, 37
Android Widget. *See* widgets
android.view, 139
android.widget, 139
AndroidManifest.xml
<uses-configuration> tag, editing, 82, 85

applications
configuration settings, defining, 92
naming, 85
versioning, 85
configuration attributes, 90-91
animated GIFs, 238
animations, 113-114
animated GIFs, 238
frame-by-frame animation, 238-239
loading, 243
sequential tweened animations, defining, 242
simultaneous tweened animations, defining, 242
storing, 98
tweened animations, 240
alpha transformations, 244
moving transformations, 245
rotating transformations, 244
scaling transformations, 244-245
antialiasing, 221
Apache HTTPClient v4, 305
APIs, 303
debug API key, obtaining, 328-329
Device Sensor, 391-394
LBS, GPS, 319-321
multimedia
audio, playing, 350-351
audio, recording, 349-350
audio, sharing, 351-352
images, assigning as wallpapers, 345
images, sharing, 344-345
ringtones, 352
still images, capturing, 339-344
video, playing, 347-348
video, recording, 346-347
telephony, 353
phone calls, placing, 363
SMS, 358-362
TelephonyManager object, 354-357
application integration
Android versus other platforms, 21-22
testing, 468
application logging, 458
application runtime environment, 26
applications
Android, developing, 27-29
billing, testing, 469
bulletproof, designing, 456
code quality, improving, 457-459
defects, handling, 459
development process, 456
feasibility testing, 456-457
mistakes, avoiding, 460
configuration settings, defining in the AndroidManifest.xml file, 92
debugging
in Android Emulator, 61-63
on hardware, 68-69
defect tracking system, developing, 461-462
directories, 253-254
reading files from, 255-256
writing files to, 254-255
input hardware, specifying, 90
installing with ADB, 528
intellectual property, protecting, 485-486
linking to other packages, 92
location-based services support, adding, 66-68
logging support, adding, 63-64
media support, adding, 64-66
permissions, registering, 88-89
preferences, 249
changing, 251-252
locating on Android file system, 252-253
private, 250
reading, 250-251
searching, 250-251
shared, 250
preparing for Android Market, 478-479
publishing, 473-474, 483
certifying the application, 477
preparing the package, 475-477
release version, testing, 477
reinstalling with ADB, 528

removing from Android Market, 483
requirements
feasibility of, 436
handset database, managing, 431–434
project requirements, 429–430
third-party requirements, incorporating, 431
use cases, developing, 430–431
running in Android Emulator, 58–59
self-distribution, 484–485
selling on other markets, 485
sharing data between, 276–277
data-binding to Gallery widget, 279
displaying images, 280–281
managedQuery() method, 278
saving images to database, 279
URIs, 278
software requirements, specifying, 90
testing
coverage, maximizing, 464–467
leveraging Android tools, 470
testing environment, managing, 462–464
uninstalling with ADB, 528
uploading to Android Market, 482–483
user interface, binding data to, 272
data adapters, 273–276
AppWidgets, 140, 215
creating, 217
hosting, 217
providers, 216
architecture
of Android mobile platform, 24
application runtime environment, 26
Linux OS, 25–26
for mobile applications, exploring, 440–441
arcs, drawing, 235
ArrayAdapter class, 206-207
ash shell
custom binaries, installing, 535–537
emulator, starting/stopping, 532
launching, 540
Monkey tool, launching, 533
shell commands, issuing, 532
shell sessions, starting, 532
SQLite databases, inspecting, 532
assessing project risks
quality assurance, 436–437
target handsets, acquiring, 435–436
target handsets, identifying, 434–435
assigning images as wallpapers, 345
AsyncTask class, 309-310
attaching debuggers to processes, 516
attributes
of AbsoluteLayout view class, 194
of FrameLayout view class, 196–197
of LinearLayout view class, 197
of RelativeLayout view class, 199–202
of TableLayout view class, 202, 204
audio
playing, 350–351
recording, 349–350
ringtones, 352
sharing, 351–352
auto-complete feature, 146-149
AutoCompleteTextView widget, 148
AVDs (Android Virtual Devices)
Android emulator, configuring, 494
creating, 495
for projects, 52, 57
targets, listing, 494–495
with custom hardware settings, 498–501
with SD card images, 498
deleting, 501
listing, 501
skins, 496–498
avoiding mistakes, 460

B

batteries, monitoring, 397-399
beginTransaction() method, 263
billing, Google checkout, 486
binding data
to AdapterView, 208
to application user interface, 272
data adapters, 273–276

bindService() method, 77
bitmaps, 227
blinking light notifications, 408-410
Bodlaender, Hans, 226
"The Brick," 11-12
BroadcastReceivers, registering, 87
broadcasts, 77
Browser Content Provider, 285
browsers, WebView, 314-317
browsing the file system, 519
bug reports, generating, 531
built-in content providers, 281
Browser, 285
CallLog, 283–284
Contacts, 286–288
MediaStore, 281–283
required permissions, 284
UserDictionary, 288
built-in layout classes, 191-193
AbsoluteLayout, 193–194
attributes, 192
FrameLayout, 195–197
LinearLayout, 197–198
RelativeLayout, 199–202
TableLayout, 202–204
built-in View container classes, 206
ArrayAdapter, 206–207
CursorAdapter, 207–208
ListActivity, 209
bulletproof applications, designing, 456
code quality, improving, 457–459
defects, handling, 459
development process, 456
feasibility testing, 456–457
mistakes, avoiding, 460
buttons, 152-154
check boxes, 154–155
radio buttons, 155–157
toggle buttons, 155

C

calculated columns, 550-552
call state, requesting, 354-356
callbacks, 78
onCreate() method, 79
onDestroy() method, 81
onPause() method, 79–81
onResume() method, 79
calling between emulator instances, 505-506
CallLog Content Provider, 283-284
CameraSurfaceView class, 340
cancel() method, 406
Canvas, 221
bitmaps, drawing/scaling, 227
capturing still images, 339-344
certifying Android applications, 477
changing preferences, 251-252
CheckBox button, 153
CheckBox widget, 154-155
Chess Utrecht font, 226
Chronometer widget, 163-165
circles, drawing, 234
clean starting state, determining on devices, 463
cleaning up OpenGL ES, 386
clearing LogCat log, 530
closing SQLite database, 269
code quality, improving, 457-459
coloring vertices, 375-376
colors, 106
command line, configuring Emulator startup options, 503
commands
ADB
directing to specific devices, 526
listing, 537
DROP TABLE, 552
comparing mobile platforms, 19
application integration, 21–22
development tools, 20–21
revenue models, 22
compiling debug code, 474
complex objects, drawing, 376-377
composite primary keys, creating tables for SQLite Student grade database, 548-549
conditionally compiling debug code, 474

configuration management systems, versioning, 439-440
configuring
Android emulator, location information, 504
development environment, 31
Intent filters, 87
locale support, 128–129
operating system for device debugging, 38–39
projects, 55–56
TextView widget
contextual links, 142–143
layout, 141–142
conformance testing, 468
connecting to a database, 540
Contacts Content Provider, 286-288
content providers, built-in, 281
Browser, 285
CallLog, 283–284
Contacts, 286–288
MediaStore, 281–283
required permissions, 284
UserDictionary, 288
Content Providers
data columns, defining, 292
data URI, defining, 292
delete() method, implementing, 296–297
getType() method, implementing, 297–298
inheriting from, 291–292
insert() method, implementing, 294–295
manifest file, updating, 298–299
query() method, implementing, 294
query() method, implementing, 293
records
adding, 289
deleting, 290–291
updating, 290
registering, 88
sharing data between applications, 276–277
data-binding to Gallery widget, 279
displaying images, 280–281
managedQuery() method, 278
saving images to database, 279
URIs, 278
update() method, implementing, 295–296
UriMatcher class, 294
content providers in OHA, 18
content-specific fees, billing for, 486
ContextMenu widget, 168-170
contextual links of TextView widget, configuring, 142-143
controlling Services, 417-418
copying files with File Explorer, 519-520
core files, 56-57
creating SD Card disk images, 498
createBitmap() method, 227
createScaledBitmap() method, 227
creating
AppWidgets, 217
AVDs, 495
for projects, 52, 57
targets, listing, 494–495
with custom hardware settings, 498–501
with SD card images, 498
files in file system, 256–257
Intents, 75
launch configuration for projects, 52, 58
layouts programmatically, 185–186
layouts with XML, 183–184
new Android projects, 55–56
NotificationManager object, 403–404
preferences, 250
Services, 413–417
SQLite database, 258–260
Cursor objects
iterating query results, 264–265
querying SQLite database, 264
CursorAdapter class, 207-208
custom binaries, installing, 535-537
custom fonts, 225
custom typefaces, 225
customizing notifications, 410-411

D

dangerous protection level, 89
data adapters, SimpleCursorAdapter, 273-276
data columns, defining for custom Content Provider, 292
databases
connecting to, 540
field names, organizing, 269-271
schema, listing with sqlite3, 541-542
DatePicker widget, 157-159
DDMS (Dalvik Debug Monitor Service), 44, 513
Eclipse DDMS Perspective, 513
Emulator Control tab, 521
GC, forcing, 517
heap activity, monitoring, 517
key features, 515
LogCat tool, 522
processes, 515
killing, 518
screen captures, taking, 523
threads, monitoring, 517
debug API key, obtaining, 328-329
debuggers, attaching to processes, 516
debugging applications
from hardware, 68-69
in Android Emulator, 61-63
default fonts, 224
default resource directories, 96
default typefaces, 224
defect tracking system, developing, 461-462
defects, handling, 459
defining
application configuration settings in AndroidManifest.xml file, 92
data columns for Content Providers, 292
data URI for Content Providers, 292
sequential tweened animations, 242
shape drawables as XML resources, 229
shape drawables programmatically, 230
simultaneous tweened animations, 242
tweened animations
as XML resources, 241
programmatic, 242
delete() method, implementing for custom Content Provider, 296-297
deleting
AVDs, 501
files with File Explorer, 520
records
from Content Providers, 290-291
in SQLite database, 262
tables, 552
from SQLite database, 268
deploying mobile applications, 443-444
designing
bulletproof applications, best practices
code quality, improving, 457-459
defects, handling, 459
development process, 456
feasibility testing, 456-457
mistakes, avoiding, 460
layouts, 120-122
mobile applications
architectures, 440-441
defect tracking system, developing, 461-462
ease of maintenance, 453-454
ease of porting, 455
ease of upgrade, 454-455
extensibility, 441-442
interoperability, 442
leveraging Android tools, 455
mistakes, avoiding, 455
profitability, 452
robust application design, 450-451
security, 451-452
testing coverage, maximizing, 464-467
testing environment, managing, 462-464
third-party standards, leveraging, 453
user demands, meeting, 448
user interfaces, 448-450
persistent databases, 269-271
smoke tests, 464
detecting
events, 171-172
focus changes, 175-176
gestures, 173-175

long clicks, 172–173
screen orientation changes, 176
touch mode changes, 170–171
developing
Android applications, 27–29
defect tracking system, 461–462
mobile applications, 442
development environment
configuring, 31
testing, 49
Device Sensor, 391-394
Dialogs, 213
DigitalClock widget, 165
dimensions, 106-107
directing ADB commands to specific devices, 526
directories, 56-57, 253, 518
browsing, 519
qualifiers, 129–131
reading files from
raw files, 255
XML files, 255–256
writing files to, 254–255
disabling debugging applications for publishing, 474
displaying
images using URIs, 280–281
log information with LogCat, 529
documentation,
for Android SDK, 41
writing, 437–438
drawables, 107-108
drawing
3D objects with OpenGL, 372
complex objects, 376–377
objects, texturing, 380–382
scenes, lighting, 378–379
vertices, 374–375
vertices, coloring, 375–376
bitmaps on Canvas, 227
on the screen, 219
Canvas, 221
onDraw() method, 219
Paint class, 221–224
shapes, 231
arcs, 235
circles, 234
ovals, 234
paths, 236
rectangles, 231
rectangles with rounded corners, 231–233
squares, 231
drill-down menus, 76
DROP TABLE command, 552
dumping database contents, 542

E

ease of maintenance, designing applications for, 453-454
ease of porting, designing applications for, 455
ease of upgrading, designing applications for, 454-455
Eclipse
AndroidManifest.xml file
application, versioning, 85
editing, 82, 85
naming, 85
Emulator startup options, configuring, 503
layouts, designing, 120, 122
perspectives, 61
resource values, setting, 99–102
Snake project, adding to workspace, 50–51
Eclipse DDMS Perspective, 513
Eclipse Development Environment for Java, installing, 32-33
editing
AndroidManifest.xml, 82, 85
strings, 103
EditText widget, 144-146
focus change, detecting, 175–176
EGL, initializing, 370-372
em measurement, 141
emulator. *See* Android Emulator
Emulator Control tab (DDMS), 521
enabling OpenGL ES communication with application thread, 383-386
enforcing content provider permissions at URI level, 89

events
detecting, 171–172
generating with Monkey tool, 533–534
gestures, detecting, 173–175
long-clicks, detecting, 172–173
example of AbsoluteLayout view class, 194-195
executing
SQL commands from sqlite3, 544
SQL scripts from files, 543
SQLite database queries, 265
joins, 266–267
rawQuery() method, 267–268
WHERE clause, 266
explicitly defined application permissions, 26
extending SQLiteOpenHelper class, 270-271
extensibility for mobile applications, designing, 441-442

F

feasibility of application requirements, determining, 436
feasibility testing, 456-457
features of DDMS, 515
fees, billing, 486
File Explorer
files, copying, 519–520
files, deleting, 520
file system
browsing, 519
directories, 518
files, creating, 256–257
files
retrieving from a device with ADB, 527
sending to a device with ADB, 527
storing, 98
filtering events, 530
finding your location with GPS, 319-321
focus changes, detecting, 175-176
folders, LiveFolders, 299-300
fonts, 224-225
forcing GC with DDMS, 517
foreign keys, creating tables for SQLite Student grade database, 548-549
format strings
accessing programmatically, 104–105
creating, 104
formation of OHA, 17-18
formatting strings, 104
frame-by-frame animation, 238-239
FrameLayout view class, 195
attributes, 196–197

G

Gallery widget, data-binding, 279
GC (Garbage Collector), running with DDMS, 517
generating
bug reports, 531
specific events with Monkey tool, 533–534
generating revenue, 486
geocoding locations, 321-325
GestureDetector class, 173
gestures, detecting, 173-175
getHeight() method, 227
getTextBounds() method, 227
getType() method, implementing for custom Content Provider, 297-298
getView() method, 279
getWidth() method, 227
GL, initializing, 372
GlobalFocusChange event, 171
GlobalLayout event, 171
GLSurfaceView class, 386-389
Google
Intents, 76
Internet model, 17
OHA, 19
content providers, 18
formation of, 17–18
manufacturers, 18
mobile operators, 18
Google checkout, billing, 486
Google Maps
Debug API key, obtaining, 328–329
map view
panning, 329
zooming, 330

mapping intents, 325
mapping views, 326–328
marking the spot, 331–336
GPS (Global Positioning System)
location, finding, 319–321
coordinates, sending, 509
GpsSatellite class, 336
gradients, 221
linear gradients, 222
radial gradients, 223
sweep gradients, 223
graphics, storing, 98

H

Handango, 485
handling
defects, 459
specialized test scenarios, 468–469
handset configurations, managing, 462-463
handset database, managing, 431, 433-434
heap activity, monitoring with DDMS, 517
Hierarchy Viewer tool, 188-189
history of mobile software, 9, 11
Motorola DynaTAC 8000X, 11–12
proprietary mobile platforms, 14–16
WAP, 12–14
hosting AppWidgets, 217
HTTP (HyperText Transfer Protocol), 303
HttpURLConnection object, 305

I

icons, assigning to applications, 85
identifying target handsets, 434-435
images
assigning as wallpapers, 345
sharing, 344–345
images, 108
accessing programmatically, 112–113
displaying from network resource, 310–312
Nine-Patch Stretchable graphics, 109–111
IMEs (Input Method Editors), 145
implementing
configuration management systems, 439–440
delete() method for Content Providers, 296–297
getType() method for Content Providers, 297–298
insert() method for Content Providers, 294–295
parcelable class, 420–423
query() method for Content Providers, 293–294
remote interfaces, 418–420
update() method for Content Providers, 295–296
importing data with sqlite3, 543
improving code quality, 457-459
incoming calls, simulating, 507
incoming SMS messages, simulating, 521
incoming voice calls, simulating, 521
indicator light notifications, 409-410
inheriting from Content Providers, 291-292
initializing
EGL, 370–372
GL, 372
input hardware, specifying for applications, 90
InputFilter objects, 149
insert() method, 261
implementing for custom Content Provider, 294–295
inserting records in SQLite database, 260
inspecting SQLite databases with ash, 532
installation testing, 469
installing
Android Plug-In for Eclipse, 35–37
Android SDK, 34–35
applications with ADB, 528
custom binaries, 535–537
Eclipse Development for Java, 32–33
JDK, 32
intellectual property, protecting, 485-486
Intents, 74-75
broadcasts, 77
creating, 75
filters, configuring, 87
Google, 76
information, passing, 76
mapping, 325
organizing with menus, 76

internationalization support, 131-132
Internet model, 17
Internet, accessing
Android network status, retrieving, 312-314
HttpURLConnection object, 305
images, displaying from network resource, 310-312
reading web data, 304
threads, using for network calls, 308-310
XML, parsing from network, 305-307
interoperability of mobile applications, designing, 442
IRIS GL, 365
isFinishing() method, 81
issuing shell commands, 532
iterating SQLite query results, 264-265
iteration, importance of, 429

J-K

Java, installing Eclipse Development Environment, 32-33
JDK, installing, 32
JOINs, 550
SQLite database queries, 266-267
key features of DDMS, 515
"killer" apps, testing for, 469
killing ADB server process, 526
killing processes with DDMS, 518

L

landscape mode, user interface design, 449-450
languages, locale support, 127-129
launch configuration, creating for projects, 52, 58
launching
activities, 74-76
ash, 540
Monkey tool, 533
Services, 77
layout widget, 140
layouts, 118-120
AbsoluteLayout view, 193
attributes, 194
example, 194-195
accessing programmatically, 122
built-in, 191-193
creating programmatically, 185-186
creating with XML, 183-184
designing, 120-122
FrameLayout view class, 195
attributes, 196-197
LinearLayout view class, 197-198
of TextView widget, configuring, 141-142
RelativeLayout view class, 199-202
TableLayout view class, 202-204
LBS (Location-Based Services)
adding support to applications, 66-68
APIs, GPS, 319-321
Google Maps
Debug API key, obtaining, 328-329
mapping intents, 325
mapping views, 326-330
marking the spot, 331-336
leveraging
ad revenue, 486
Android tools
for application design, 455
for application testing, 470
license agreement for Android SDK, 39-41
third-party standards
for Android testing, 467
for application design, 453
lifecycle of activities, 77
callbacks, 78
onCreate() method, 79
onDestroy() method, 81
onPause() method, 79-81
onResume() method, 79
lighting scenes, 378-379
limitations
of SQLite, 545
of emulator, 512
linear gradients, 222
LinearLayout view class, 197-198
linking applications to other packages, 92
Linux OS, 25-26
ListActivity class, 209

listing
ADB command, 537
available databases with sqlite3, 541
available tabes with sqlite3, 541
AVDs, 501
connected devices with ADB, 525
database schema with sqlite3, 541–542
table indices with sqlite3, 541
LiveFolders, 299-300
loading animations, 243
locale support, 127-129
locating preferences on Android file system, 252-253
locations, geocoding, 321- 325
LogCat, 522, 529
events, filtering, 530
log information, displaying, 529
log output, redirecting to a file, 531
log, clearing, 530
logging modes, changing, 529
secondary logs, accessing, 531
logging support, adding to Android applications, 63-64
long clicks, detecting, 172-173

M

main menu, 76
main.xml layout file, 119
making phone calls, 363
managedQuery() method, 278
managing testing environment, 462-464
manifest file, updating for Content Providers, 298-299
manipulating power settings, 510
manufacturers in OHA, 18
map view
panning, 329
zooming, 330
MapController object, 327
mapping intents, 325
mapping views, 326-328
MapView object, 327
Matrix class, transforming bitmaps, 227
The Matrix Phone, 13
maximizing testing coverage, 464-467
measureText() method, 227
media support, adding to media applications, 64-66
MediaStore Content Provider, 281, 283
menus, 114-115
Intents, organizing, 76
storing, 98
methods
addOnTouchModeChangeListener(), 171
addView(), 188
beginTransaction(), 263
bindService(), 77
cancel(), 406
createBitmap(), 227
createScaledBitmap(), 227
getHeight(), 227
getTextBounds(), 227
getView(), 279
getWidth(), 227
insert(), 261
isFinishing(), 81
managedQuery(), 278
measureText(), 227
notify(), 404–406
onCreate(), 79, 185
onDestroy(), 81
onDraw(), 219
onOptionsMenuSelected(), 168
onPause(), 79–81
onResume(), 79
onSaveInstanceState(), 81
onTouchEvent(), 173
rawQuery(), 267–268
recycle(), 228
remove(), 262
setColor(), 221
setContentView(), 184–186
setFilters(), 149
setOneShot(), 239
setText(), 185
setTheme(), 179
startActivity(), 74
startService(), 77
mimicking real-world activities in testing environment, 464

minimum Android SDK version, specifying, 91-92
mistakes, avoiding, 460
during application testing, 470
in application design, 455
MobiHand, 485
mobile application servers, testing, 466
mobile applications. *See also* mobile platforms
architectures, 440-441
deploying, 443-444
designing, best practices, 440
ease of maintenance, 453-454
ease of porting, 455
ease of upgrading, 454-455
mistakes, avoiding, 455
profitability, 452
robust design, 450-451
security, 451-452
third-party standards, leveraging, 453
user demands, meeting, 448
user interface design, 448-450
developing, 442
extensibility, designing, 441-442
interoperability, designing, 442
supporting, 444
testing, 442-443
mobile development process, 427
application requirements
feasibility, 436
handset database, managing, 431-434
project requirements, 429-430
third-party requirements, incorporating, 431
use cases, 430-431
configuration management systems, implementing, 439-440
project documentation, 437
test plans, 438
third-party documentation, 438
project risks, assessing, 434-436
quality assurance, 436-437
software methodology
iteration, 429
waterfall approaches, limitations of, 428
mobile operators in OHA, 18
mobile platforms
Android
applications, developing, 27-29
architecture, 24-26
as emerging technology, 23
security, 26-27
application integration, comparing, 21-22
comparing, 19-20
development tools, comparing, 20-21
revenue models, comparing, 22
mobile software, history of, 9, 11
Motorola DynaTAC 8000X, 11-12
proprietary mobile platforms, 14-16
WAP, 12-14
monitoring
batteries, 397-399
events with Monkey tool, 533
heap activity with DDMS, 517
network status, 510
threads with DDMS, 517
Monkey tool
events
generating, 533-534
monitoring, 533
launching, 533
seed option, 535
throttle option, 535
Motorola DynaTAC 8000X, 11-12
Motorola StarTAC, 12
moving transformations, 245
multimedia
audio
playing, 350-351
recording, 349-350
sharing, 351-352
images
assigning as wallpapers, 345
sharing, 344-345
ringtones, 352
still images, capturing, 339-344
video
playing, 347-348
recording, 346-347

"MultiNational" application, 131-132
multipart text messages, 361

N

naming applications
for publishing, 474
with AndroidManifest.xml file, 85
naming conventions for strings, 98
network status, monitoring, 510
Nine-Patch Stretchable graphics, 109-111
noise notifications, 410
normal protection level, 89
NotificationManager object, creating, 403-404
notifications, 403-407
blinking lights, 408-410
customizing, 410-411
noise, 410
vibration, 407-408
notify() method, 404-406

O

objects
Dialog, 213
texturing, 380-382
obtaining
debut API key, 328-329
WiFi information, 395-397
OHA (Open Handset Alliance), 19
content providers, 18
formation of, 17-18
manufacturers, 18
mobile operators, 18
onCreate() method, 79, 185
onDestroy() method, 81
onDraw() method, 219
onOptionsItemSelected() method, 168
onPause() method, 79-81
onResume() method, 79
onSaveInstanceState() method, 81
onTouchEvent() method, 173
OpenGL ES
3D applications, 366
drawing, 372, 374-382
EGL, initializing, 370-372
GL, initializing, 372
OpenGL ES thread, starting, 369-370
SurfaceView, creating, 367-368
cleaning up, 386
communication with application thread, enabling, 383-386
GLSurfaceView class, 386-389
operating system, configuring for device debugging, 38-39
OptionsMenu widget, 166-168
organizing
database field names, 269-271
Intents with menus, 76
resources, 134-135
OrientationListener class, 176
outsourcing testing responsibilities, 471
ovals, drawing, 234

P

Paint class
antialiasing property, 221
gradients, 221
linear gradients, 222
radial gradients, 223
sweep gradients, 223
setColor() method, 221
setFlags() method, 225
styles, 221
utilities, 224
panning map view, 329
parcelable class, implementing, 420-423
passing information with Intents, 76
paths, drawing, 236
performance testing, 469
permissions
on Android applications, 26
registering, 88-89
requirements for content providers, 284
persistent databases, designing, 269-271
perspectives, 61
phone calls, making, 363
phone numbers, 356-357
phone state information, gaining access permission, 354

placing phone calls, 363
playing
audio, 350–351
video, 347–348
plugins, installing Android Plug-In for Eclipse, 35-37
portrait mode, user interface design, 449-450
power settings, manipulating, 510
PreDraw event, 171
preferences, 249
changing, 251–252
locating on Android file system, 252–253
private, 250
reading, 250–251
searching, 250–251
shared, 250
preparing
applications for Android Market, 478–479
packages for publication, 475–477
private preferences, 250
processes, 515
debuggers, attaching, 516
killing with DDMS, 518
product internationalization, testing, 468
profitable mobile application design, 452
programmatic access
images, 112–113
layouts, 122
resources, 99
strings, 104–105
styles, 126
ProgressBar widget, 159-161
projects
AVDs, creating, 57
creating, 55–56
documentation, writing, 437–438
launch configuration, creating, 58
requirements, determining, 429–430
risks, assessing, 434–437
proprietary mobile platforms, 14-16
protecting intellectual property, 485-486
publishing applications
certifying the application, 477
debugging, disabling, 474
naming the application, 474
on Android Market, 483
preparing the package, 475–477
release version, testing, 477
versioning the application, 474
Pull Parsers, 307

Q

qualifiers, 129-131
quality assurance risks, 436-437
query() method, implementing for custom Content Provider, 293-294
querying SQLite database, 265
cursors, 263–264
iterating query results, 264–265
joins, 266–267
rawQuery() method, 267–268
WHERE clause, 266

R

radial gradients, 223
radio buttons, 155-157
RadioButton, 153
RatingBar widget, 162-163
raw files, 117
reading from default application directory, 255
rawQuery() method, SQLite database queries, 267-268
reading
files in application directory
raw files, 255
XML files, 255–256
preferences, 250–251
web data, 304
real-world activities, mimicking, 464
receiving SMS, 360-362
recording
audio, 349–350
video, 346–347

records
adding to Content Providers, 289
deleting
from Content Providers, 290-291
in SQLite database, 262
inserting in SQLite database, 260
updating
in Content Providers, 290
SQLite database, 261
rectangles
drawing, 231
with rounded corners, drawing, 231-233
recurring fees, billing for, 486
recycle() method, 228
redirecting LogCat log output to a file, 531
referencing
resources, 117-118
system resources, 126-127
refund policy for Android Market, 483
regions, locale support, 127-129
registering
activities, 86-87
BroadcastReceivers, 87
content providers, 88
permissions, 88-89
services, 87
reinstalling applications with ADB, 528
RelativeLayout view class, 199-202
release version, testing, 477
remote interfaces, implementing, 418-420
remove() method, 262
removing applications from Android Market, 483
reporting problems with Android SDK, 38
requesting
call state, 354-356
service information, 356
resource directory qualifiers, 129-131
resource hierarchy directory, 96
ResourceRoundup project, setting resource values example, 99-102
resources
animations, 113-114
storing, 98
colors, 106
dimensions, 106-107
drawables, 107-108
files, storing, 98
graphics, storing, 98
images, 108
Nine-Patch Stretchable graphics, 109-111
layouts, 118-120
designing, 120-122
menus, 114-115
storing, 98
naming conventions, 98
organizing, 134-135
programmatic access, 99
raw files, 117
referencing, 117-118
storing, 95-96
string arrays, 105-106
strings
accessing programmatically, 104-105
editing, 103
formatting, 104
styles, 123-124
accessing programmatically, 126
system resources, referencing, 126-127
themes, 126
value types, 96-98
values, setting, 99-102
XML files, 115-116
retrieving files from a device with ADB, 527
revenue models, Android versus other platforms, 22
RGB color values, 106
ringtones, 352
robust mobile application design, 450-451
rotating transformations, 244

S

sample applications, 47
scaling
bitmap graphics, 227
transformations, 244-245
scenes, lighting, 378-379

schema for SQLite Student grade database, example, 546
screen captures, taking with DDMS, 523
screen orientation changes, detecting, 176
ScrollView widget, 213
SD Card disk images, creating, 498
searching preferences, 250-251
secure mobile application design, 451-452
security of Android mobile platform
permissions, 26
trust relationships, 27
seed option (Monkey tool), 535
SeekBar widget, 161-162
selection events, handling, 209
self-distribution, 484-485
selling applications
on other markets, 485
on your own server, 484-485
sending
files to a device with ADB, 527
GPS coordinates, 509
SMS, 358-359
sqlite output to a file, 542
SMS messages between Emulator instances, 507
sequential tweened animations, defining, 242
server process (ADB), starting/stopping, 526
service information, requesting, 356
services, 77
controlling, 417-418
creating, 413-417
launching, 77
registering, 87
setColor() method, 221
setContentView() method, 184-186
setFilters() method, 149
setFlags() method, 225
setLatestEventInfo() method, 411
setOneShot() method, 239
setText() method, 185
setTheme() method, 179
shape drawables
defining as XML resouces, 229
defining programmatically, 230
ShapeDrawable class, 229
shapes, drawing, 229-231
arcs, 235
circles, 234
ovals, 234
paths, 236
rectangles, 231-233
squares, 231
Shared Preferences, 249-250
SharedPreferences.Editor interface, methods, 251-252
sharing
audio, 351-352
data between applications, content providers, 276-277
data-binding to Gallery widget, 279
displaying images, 280-281
managedQuery() method, 278
saving images to database, 279
URIs, 278
images, 344-345
sharing shell command, issuing, 532
shell sessions, starting, 532
signature protection level, 89
SimpleCursorAdapter, 273-276
simulating
incoming SMS messages, 521
incoming calls, 507, 521
SMS messages, 508-509
simultaneous tweened animations, defining, 242
skins (AVDs), 496-498
SlideME, 485
SlidingDrawer widget, 215
smoke tests, designing, 464
SMS (Short Message Service), 353, 358
phone calls, placing, 363
messages
receiving, 360-362
sending, 358-359, 507
simulating, 508-509
Snake project
adding to Eclipse workspace, 50-51
AVD, creating, 52

launch configuration, creating, 52
running in Android Emulator, 54
software development methodology
iteration, importance of, 429
waterfall approaches, limitations of, 428
software requirements, specifying for applications, 90
specialized test scenarios, handling, 468-469
Spinner widgets, 150-152
SQL (Structured Query Language)
commands, executing from sqlite3, 544
scripts, executing from files, 543
SQLite, 539
application user interface, binding data to, 272-276
closing, 269
creating, 258-260
inspecting with ash, 532
limitations of, 545
persistent databases, designing, 269-271
queries
cursors, 263-265
executing, 265-268
records
deleting, 262
inserting, 260
updating, 261
structured data, storing, 257-258
Student grade database example
calculated columns, 550-552
JOINs, 550
schema, 546
tables, creating, 546
tables, inserting data into, 547
tables, altering data in, 549
tables, creating, 548-549
tables, querying, 547-548
tables, deleting, 268
transactions, 262-263
sqlite3, 540-541
available databases, listing, 541
available tables, listing, 541
data, importing, 543
database contents, dumping, 542
database schema, listing, 541-542
output, sending to a file, 542
SQL
commands, executing, 544
scripts, executing from files, 543
table indices, listing, 541
SQLiteOpenHelper class, extending, 270-271
SQLiteQueryBuilder object, 267
squares, drawing, 231
SSL (Secure Sockets Layer), 303
startActivity() method, 74
starting
activities, 74-76
ADB server process, 526
emulator with ash, 532
OpenGL ES thread, 369-370
Services, 77
shell sessions, 532
startService() method, 77
startup options (Android emulator), 502
configuring from command line, 503
configuring with Eclipse, 503
still images, capturing, 339-344
stopping
ADB server process, 526
emulator with ash, 532
storing
resources, 95
animations, 98
files, 98
graphics, 98
menus, 98
naming conventions, 98
resource directory hierarchy, 96
strings, 98
value types, 96-98
structured data with SQLite, 257-258
string arrays, 105-106
strings
accessing programmatically, 104-105
editing, 103
formatting, 104
storing, 98
structured data, storing with SQLite, 257-258

Student grade database, SQLite example
calculated columns, 550–552
JOINs, 550
schema, 546
tables, altering data in, 549
tables, creating, 546–549
tables, inserting data into, 547
tables, querying, 547–548
styles, 123-124, 177-178
accessing programmatically, 126
themes, 179–180
subclasses of ViewGroup, 188
supporting mobile applications, 444
SurfaceView, creating for 3D applications, 367-368
sweep gradients, 223
switchers, 212
system resources, referencing, 126-127

T

tabbed interface (Android SDK), 210-212
TableLayout view class, 202-204
tables
altering data for SQLite Student grade database example, 549
creating for SQLite Student grade database example, 546–549
deleting, 552
from SQLite database, 268
indices, listing with sqlite3, 541
inserting data into for SQLite Student grade database example, 547
JOINs, Student grade database example, 550
calculated columns, 550–552
listing with sqlite3, 541
querying for SQLite Student grade database example, 547–548
taking screen captures with DDMS, 523
target handsets
acquiring, 435–436
identifying, 434–435
telephony APIs, 353
phone calls, placing, 363
SMS, 358
receiving, 360–362
sending, 358–359
TelephonyManager object
call state, requesting, 354–356
phone numbers, 356–357
service information, requesting, 356
TelephonyManager object
call state, requesting, 354–356
phone numbers, 356–357
service information, requesting, 356
test plans, writing, 438
testing
applications, 470
coverage, maximizing, 464–467
development environment, 49
environment, managing, 462–464
mobile applications, 442–443
outsourcing, 471
testing release version, 477
text messaging, multipart text messages, 361
texturing objects, 380-382
TextView widget, 140
contextual links, configuring, 142–143
layout, configuring, 141–142
themes, 126, 179-180
third-party application requirements, incorporating, 431
third-party documentation, writing, 438
third-party standards, leveraging, 453, 467
threads
monitoring with DDMS, 517
OpenGL ES
communication with application thread, enabling, 383–386
starting, 369–370
using for network calls, 308–310
throttle option (Monkey tool), 535
TimePicker widget, 159
ToggleButton widget, 153-155
tools
ADB, 45
Android Emulator, 43–44
Android Hierarchy Viewer, 46
DDMS, 44
Hierarchy Viewer, 188–189

touch model, detecting changes, 170-171
transactions, 262-263
transformations
alpha transformations, 244
moving transformations, 245
rotating transformations, 244
scaling transformations, 244-245
tweening, 240
transforming bitmaps with Matrix class, 227
transitions, 81-82
trust relationships, 27
tweened animations, 240
alpha transformations, 244
defining as XML resources, 241
defining programmatically, 242
moving transformations, 245
rotating transformations, 244
scaling transformations, 244-245
tweening, 113
typefaces, 224-225

U

uninstalling applications with ADB, 528
unit tests, developing, 458-459
update() method, implementing for custom Content Provider, 295-296
updating
manifest files for Content Providers, 298-299
records
in Content Providers, 290
in SQLite database, 261
table data, SQLite Student grade database, 549
upgrades, testing, 468
upgrading Android SDK, 37
uploading applications to Android Market, 482-483
UriMatcher class, 294
URIs
defining for custom Content Provider, 292
images, displaying, 280-281
locating content, 278
usability testing, 467
use cases, developing, 430-431
user demands, meeting, 448
user interface elements
AnalogClock widget, 165
auto-complete feature, 146-149
buttons, 152-154
check boxes, 154-155
radio buttons, 155-157
toggle buttons, 155
Chronometer widget, 163-165
ContextMenu widget, 168-170
DatePicker widget, 157-159
designing, 448-450
DigitalClock widget, 165
EditText widget, 144-146
InputFilter objects, 149
layout with ViewGroups, 188
Hierarchy Viewer, 188-191
OptionsMenu widget, 166-168
ProgressBar widget, 159-161
RatingBar widget, 162-163
SeekBar widget, 161-162
Spinner widgets, 150-152
tabbed interface, 210-212
TextView widget, 140
contextual links, configuring, 142-143
layout, configuring, 141-142
TimePicker widget, 159
UserDictionary Content Provider, 288

V

version system, implementing, 439-440
versioning applications
for publishing, 474
with AndroidManifest.xml file, 85
vertices
coloring, 375-376
drawing, 374-375
vibrate functionality, 407-408
video
playing, 347-348
recording, 346-347
View class, 139

view containers
built-in, 206
ArrayAdapter, 206-207
CursorAdapter, 207-208
ListActivity, 209
Dialogs, 213
ScrollView, 213
SlidingDrawer, 215
switchers, 212
ViewGroups, 187
as View containers, 188
for layout, 188
views, mapping, 326-328
ViewSwitcher widget, 212
visual appeal, testing, 467

W

wallpapers, assigning images as, 345
WAP, 12-14
waterfall development methods, limitations of, 428
WebView, browsing the web, 314-317
WHERE clause, SQLite database queries, 266
widgets, 139
AdapterView
ArrayAdapter, 206-207
binding data to, 208
CursorAdapter, 207-208
ListActivity, 209
selection events, handling, 209
AnalogClock, 165
AutoCompleteTextView, 148
binding data to, SimpleCursorAdapter, 273-274, 276
Buttons, 153
check boxes, 154-155
radio buttons, 155-157
toggle buttons, 155
Chronometer, 163-165
ContextMenu, 168-170
DatePicker, 157-159
DigitalClock, 165
EditText, 144-146
focus changes, detecting, 175-176
Gallery, data-binding, 279
layout, 140
OptionsMenu, 166-168
ProgressBar, 159-161
RatingBar, 162-163
ScrollView, 213
SeekBar, 161-162
SlidingDrawer, 215
Spinner, 150-152
TextView, 140
contextual links, configuring, 142-143
layout, configuring, 141-142
TimePicker, 159
ViewSwitcher, 212
WebView, browsing the web, 314-317
WiFi information, obtaining, 395-397
working with phone numbers, 356-357
"Working Square" principle, 449
writing
files to application directory, 254-255
project documentation, 437-438

X-Y-Z

XML (eXtensible Markup Language)
files, 115-116
reading from default application directory, 255-256
layouts, creating, 183-184
parsing from network, 305-307
Pull Parsers, 307
zooming map view, 330