



LEARNING **Android™** APPLICATION PROGRAMMING FOR THE **Kindle Fire™**

A Hands-On Guide to Building Your First Android Application



LAUREN DARCEY
SHANE CONDER

www.it-ebooks.info

Praise for *Learning Android Application Programming for the Kindle Fire*



“Now is a great time to learn how to program for the Kindle Fire, and this book is the perfect companion for your journey! Distilled within the text are the proven techniques, best practices, and hard-won wisdom of two of the mobile industry’s leading pioneers. You’ll be programming Kindle Fire apps in no time!”

—**Mark Hammonds**, Mobile Engineer, and Managing Editor, Mobiletuts+

“Learning Android Application Programming for the Kindle Fire is a must-have developers’ resource specific to the Kindle Fire. This book takes you from SDK installation to APK publication with lots of examples and tips along the way.”

—**Jim Hathaway**, Web Developer

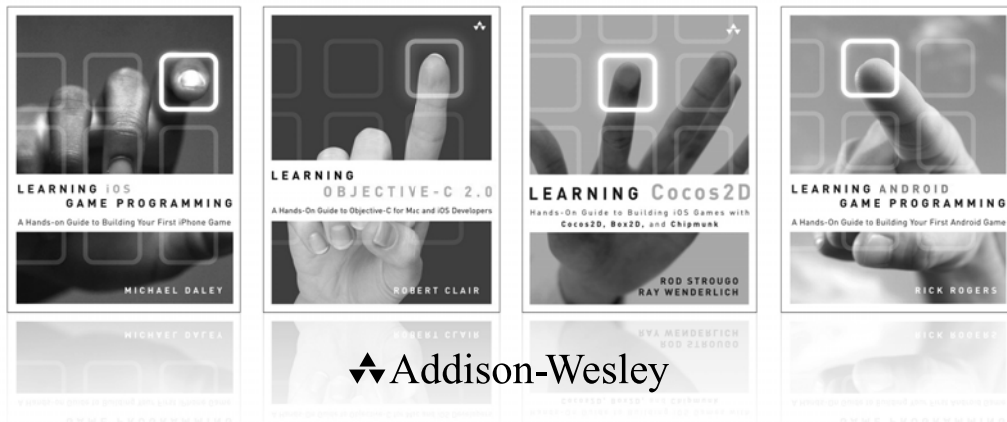
“If you want to bring your application to Amazon’s exciting new Android-based tablet, look no further than this book. Darcey and Conder show you how to go from idea to application for the Kindle Fire quickly and clearly with a sample application that shows you everything you’ll need to build your application. Along the way, you’ll pick up valuable tips on developing software in general.”

—**Ray Rischpater**, Senior Research Engineer, Nokia Research Center

“I used Lauren Darcey and Shane Conder’s books to start developing Android apps fast. I knew iOS development but needed to learn the basics of Android development. This was a great book to show me how, and I started coding the same day.”

—**Scott Walker**, Developer

Addison-Wesley Learning Series



Visit informit.com/learningseries for a complete list of available publications.

The Addison-Wesley Learning Series is a collection of hands-on programming guides that help you quickly learn a new technology or language so you can apply what you've learned right away.

Each title comes with sample code for the application or applications built in the text. This code is fully annotated and can be reused in your own projects with no strings attached. Many chapters end with a series of exercises to encourage you to reexamine what you have just learned, and to tweak or adjust the code as a way of learning.

Titles in this series take a simple approach: they get you going right away and leave you with the ability to walk off and build your own application and apply the language or technology to whatever you are working on.

◆◆ Addison-Wesley

informIT.com

Safari[®]
Books Online

Learning Android™ Application Programming for the Kindle Fire™

This page intentionally left blank

Learning Android™ Application Programming for the Kindle Fire™

A Hands-On Guide to Building
Your First Android Application

Lauren Darcey
Shane Conder

◆ Addison-Wesley

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

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

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

Android™ is a trademark of Google, Inc. Kindle Fire™ is a trademark of Amazon.com, Inc.

Android™ and Google® are trademarks of Google, Inc. Kindle Fire™ and Amazon.com® are trademarks of Amazon.com, Inc. Neither Amazon.com, Inc., nor Google, Inc., have authorized or approved publication of this work and references to their marks herein are not intended to imply their sponsorship or affiliation with this work.

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

The Library of Congress cataloging-in-publication data is on file.

Copyright © 2012 Lauren Darcey and Shane Conder

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

ISBN-13: 978-0-321-83397-6

ISBN-10: 0-321-83397-X

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.

First printing: July 2012

Editor-in-Chief
Mark Taub

Executive Editor
Laura Lewin

Development Editor
Sheri Cain

Managing Editor
Kristy Hart

Project Editor
Jovana San
Nicolas-Shirley

Indexer
Larry Sweazy

Proofreader
Gill Editorial
Services

Technical Reviewers
Jim Hathaway
Ray Rischpater
Mike Wallace

Publishing Coordinator
Olivia Basegio

Cover Designer
Chuti Prasertsith

Compositor
Nonie Ratcliff



For Ellie



Contents

Introduction 1

Part I: Kindle Fire Fundamentals

- 1 Getting Started with Kindle Fire 9
- 2 Mastering the Android Development Tools 29
- 3 Building Kindle Fire Applications 39
- 4 Managing Application Resources 53
- 5 Configuring the Android Manifest File 69
- 6 Designing an Application Framework 83

Part II: Building an Application Framework

- 7 Implementing an Animated Splash Screen 101
- 8 Implementing the Main Menu Screen 117
- 9 Developing the Help and Scores Screens 133
- 10 Collecting User Input 149
- 11 Using Dialogs to Collect User Input 165
- 12 Adding Application Logic 181
- 13 Adding Network Support 199
- 14 Exploring the Amazon Web Services SDK for Android 225

Part III: Publishing Your Kindle Fire Application

- 15 Managing Alternative and Localized Resources 233

- 16 Testing Kindle Fire Applications 249**
- 17 Registering as an Amazon Application Developer 261**
- 18 Publishing Applications on the Amazon Appstore 271**

Part IV: Appendixes

- A Configuring Your Android Development Environment 279**
- B Eclipse IDE Tips and Tricks 289**
- C Supplementary Materials 299**

Table of Contents

Introduction 1

I: Kindle Fire Fundamentals

1 Getting Started with Kindle Fire 9

Introducing Android 9

Google and the Open Handset Alliance 9

Android Makes Its Entrance 10

Cheap and Easy Development 10

Familiarizing Yourself with Eclipse 12

Creating Android Projects 13

Exploring Your Android Project Files 16

Editing Project Resources 17

Running and Debugging Applications 20

Managing Android Virtual Devices 20

Creating Debug and Run Configurations in Eclipse 22

Launching Android Applications Using the Emulator 22

Debugging Android Applications Using DDMS 25

Launching Android Applications on a Device 25

Summary 28

Exercises 28

2 Mastering the Android Development Tools 29

Using the Android Documentation 29

Debugging Applications with DDMS 31

Managing Tasks 32

Browsing the Android File System 33

Taking Screenshots of the Emulator or Device 34

Viewing Log Information 35

Working with the Android Emulator 36

Providing Input to the Emulator 36

Using Other Android Tools 36

Summary 36

3 Building Kindle Fire Applications	39
Designing an Android Application	39
Designing Application Features	40
Determining Application Activity Requirements	40
Implementing Application Functionality	41
Using the Application Context	42
Retrieving Application Resources	42
Accessing Application Preferences	42
Accessing Other Application Functionality Using Contexts	43
Working with Activities	43
Launching Activities	43
Managing Activity State	44
Shutting Down Activities	45
Working with Intents	46
Passing Information with Intents	47
Using Intents to Launch Other Applications	47
Working with Dialogs	48
Working with Fragments	49
Logging Application Information	50
Summary	51
Exercises	51
 4 Managing Application Resources	 53
Using Application and System Resources	53
Working with Application Resources	53
Working with System Resources	56
Working with Simple Resource Values	57
Working with Strings	57
Working with Colors	57
Working with Dimensions	58
Working with Drawable Resources	59
Working with Images	59
Working with Other Types of Drawables	60
Working with Layouts	60
Designing Layouts Using the Layout Resource Editor	61
Designing Layouts Using XML	62

Working with Files	64
Working with XML Files	64
Working with Raw Files	65
Working with Other Types of Resources	66
Summary	66
Exercises	66
5 Configuring the Android Manifest File	69
Exploring the Android Manifest File	69
Using the Manifest Tab	70
Using the Application Tab	71
Using the Permissions Tab	71
Using the Instrumentation Tab	72
Using the AndroidManifest.xml Tab	73
Configuring Basic Application Settings	73
Naming Android Packages	74
Versioning an Application	74
Setting the Minimum Android API Version	75
Naming an Application	76
Providing an Icon for an Application	76
Providing an Application Description	76
Setting Debug Information for an Application	77
Setting Other Application Attributes	77
Defining Activities	77
Registering Activities	77
Designating the Launch Activity	78
Managing Application Permissions	79
Managing Other Application Settings	81
Summary	81
Exercises	81
6 Designing an Application Framework	83
Designing an Android Trivia Game	83
Determining High-Level Game Features	83
Determining Activity Requirements	84
Determining Screen-Specific Game Features	85

Implementing an Application Prototype	90
Reviewing the Accompanying Source Code	90
Creating a New Android Project	90
Adding Project Resources	91
Implementing Application Activities	92
Creating Application Preferences	93
Running the Game Prototype	95
Creating a Debug Configuration	95
Launching the Prototype in the Emulator	95
Exploring the Prototype Installation	96
Summary	97
Exercises	97

II: Building an Application Framework

7 Implementing an Animated Splash Screen 101

Designing the Splash Screen	101
Implementing the Splash Screen Layout	102
Adding New Project Resources	103
Updating the Splash Screen Layout	106
Working with Animation	110
Adding Animation Resources	110
Animating Specific Views	112
Setting the Image Animations	113
Handling Animation Lifecycle Events	114
Summary	115
Exercises	116

8 Implementing the Main Menu Screen 117

Designing the Main Menu Screen	117
Determining Main Menu Screen Layout Requirements	118
Designing the Screen Header	118
Designing the GridView Control	118
Finishing Touches for the Main Menu Layout Design	119
Implementing the Main Menu Screen Layout	119
Adding New Project Resources	120
Updating the Main Menu Screen Layout Files	121

Working with the GridView Control	124
Filling a GridView Control	124
Listening for GridView Events	127
Working with Other Menu Types	128
Adding an Options Menu to the Game Screen	129
Summary	131
Exercises	131
9 Developing the Help and Scores Screens	133
Designing the Help Screen	133
Implementing the Help Screen Layout	135
Adding New Project Resources	135
Updating the Help Screen Layout	135
Working with Files	136
Adding Raw Resource Files	136
Accessing Raw File Resources	137
Designing the Scores Screen	138
Determining Scores Screen Layout Requirements	138
Adding the TabHost Control	139
Implementing the Scores Screen Layout	141
Adding New Project Resources	141
Updating the Scores Screen Layout	142
Building a Screen with Tabs	144
Configuring the TabHost Control	144
Adding Tabs to the TabHost Control	145
Setting the Default Tab	145
Working with XML	146
Retrieving XML Resources	146
Parsing XML Files with XmlResourceParser	146
Applying Finishing Touches to the Scores Screen	147
Summary	148
Exercises	148
10 Collecting User Input	149
Designing the Settings Screen	149
Implementing the Settings Screen Layout	151
Adding New Project Resources	151
Updating the Settings Screen Layout	154

Using Common Form Controls	155
Working with <code>EditText</code> Controls	156
Working with <code>Spinner</code> Controls	159
Saving Form Data with <code>SharedPreferences</code>	161
Defining <code>SharedPreferences</code> Entries	161
Saving Settings to <code>SharedPreferences</code>	161
Reading Settings from <code>SharedPreferences</code>	162
Summary	163

11 Using Dialogs to Collect User Input 165

Working with <code>Activity</code> Dialogs	165
Exploring the Different Types of Dialogs	166
Tracing the Lifecycle of a Dialog	167
Using the <code>DatePickerDialog</code> Class	168
Adding a <code>DatePickerDialog</code> to a Class	168
Initializing a <code>DatePickerDialog</code>	170
Launching <code>DatePickerDialog</code>	170
Working with Custom Dialogs	171
Adding a Custom Dialog to the Settings Screen	172
Summary	178
Exercises	178

12 Adding Application Logic 181

Designing the Game Screen	181
Implementing the Game Screen Layout	183
Adding New Project Resources	184
Updating the Game Screen Layout	185
Working with <code>ViewSwitcher</code> Controls	186
Initializing Switcher Controls	187
Implementing Switcher Factory Classes	187
Updating the <code>TextSwitcher</code> Control	189
Updating the <code>ImageSwitcher</code> Control	189
Wiring Up Game Logic	190
Adding Game State Settings to the <code>SharedPreferences</code>	191
Retrieving, Parsing, and Storing Book Data	192
Summary	197
Exercises	197

13 Adding Network Support	199
Designing Network Applications	199
Working with an Application Server	199
Managing Lengthy Network Operations	200
Informing the User of Network Activity	201
Developing Network Applications	201
Enabling Network Testing on the Emulator	202
Testing Network Applications on Kindle Fire	202
Accessing Network Services	202
Planning Have You Read That? Network Support	202
Setting Network Permissions	203
Checking Network Status	203
Using HTTP Networking	203
Indicating Network Activity with Progress Bars	204
Displaying Indeterminate Progress	204
Displaying Determinate Progress	204
Displaying Progress Dialogs	204
Running Tasks Asynchronously	206
Using <code>AsyncTask</code>	206
Using Threads and Handlers	207
Downloading and Displaying Score Data	207
Extending <code>AsyncTask</code> for Score Downloads	207
Starting the Progress Indicator with <code>onPreExecute()</code>	208
Clearing the Progress Indicator with <code>onPostExecute()</code>	209
Handling Cancellation with <code>onCancelled()</code>	209
Handling Processing with <code>doInBackground()</code>	210
Handling Progress Updates with <code>onProgressUpdate()</code>	211
Starting the <code>ScoreDownloaderTask</code>	212
Downloading and Parsing Batches of Books	213
Extending <code>AsyncTask</code> for Book Downloads	213
Starting the Progress Dialog with <code>onPreExecute()</code>	214
Dismissing the Progress Dialog with <code>onPostExecute()</code>	214
Handling the Background Processing	215
Starting the <code>BookListDownloaderTask</code>	215
Determining What Data to Send to the Server	216
Keeping Player Data in Sync	216

Uploading Settings Data to a Remote Server	217
Working with Android Services	218
Implementing UploadTask	220
Uploading Player Data with the HTTP GET Method	220
Uploading Score Data to a Remote Server	223
Downloading Friends' Score Data	224
Summary	224
Exercises	224

14 Exploring the Amazon Web Services SDK for Android 225

The 10,000-Foot View of AWS	225
Exploring the AWS Offerings	226
Using AWS Database and Storage Services	227
Using AWS Messaging and Notification Services	227
Using AWS Infrastructure and Administrative Services	228
Summary	228
Exercises	228

III: Publishing Your Kindle Fire Application

15 Managing Alternative and Localized Resources 233

Using the Alternative Resource Hierarchy	233
Understanding How Resources Are Resolved	234
Organizing Alternative Resources with Qualifiers	234
Using Alternative Resources Programmatically	236
Organizing Application Resources Efficiently	236
Customizing the Application Experience	237
Updating the Main Screen	237
Updating the Splash Screen	238
Updating the Game Screen	239
Updating the Other Screens	240
Internationalizing Android Applications	240
How Android Localization Works	241
How the Android Operating System Handles Locale	241
How Applications Handle Locales	241
How Kindle Fire Handles Locales	243

Android Internationalization Strategies	243
Forgoing Application Internationalization	244
Limiting Application Internationalization	245
Implementing Full Application Internationalization	245
Using Localization Utilities	246
Determining System Locale	246
Formatting Strings Like Dates and Times	246
Handling Currencies	247
Summary	247
Exercises	247
16 Testing Kindle Fire Applications	249
Testing Best Practices	249
Developing Coding Standards	250
Performing Regular Versioned Builds	250
Using a Defect Tracking System	251
Developing Good Test Plans	251
Maximizing Test Coverage	252
Testing on the Emulator	252
Testing on Target Devices	253
Performing Automated Testing	253
Summary	260
Exercises	260
17 Registering as an Amazon Application Developer	261
Understanding the Release Process	261
Preparing the Release Candidate Build	263
Preparing the Android Manifest File for Release	263
Protecting Your Application from Software Pirates	264
Readying Related Services for Release	265
Testing the Application Release Candidate	265
Signing Up as an Amazon App Developer	266
Packaging and Signing an Application	267
Digitally Signing Applications	267
Exporting and Signing the Package File	267

Testing the Signed Application Package	268
Installing the Signed Application Package	269
Verifying the Signed Application	269
Summary	270
Exercises	270

18 Publishing Applications on the Amazon Appstore 271

Selling on the Amazon Appstore	271
Signing Up for a Developer Account	271
Complying with the Developer License Agreement	272
Uploading an Application	272
Understanding Amazon Appstore Royalties	273
Using Other Developer Account Benefits	273
Generating Reports	274
Summary	274
Exercises	274

IV: Appendixes

A Configuring Your Android Development Environment 279

Configuring Your Development Environment	279
Development Machine Prerequisites	280
Installing the Java Development Kit	281
Installing the Eclipse IDE	281
Installing the Android SDK	281
Installing and Configuring the Android Plug-In for Eclipse (ADT)	282
Downloading Android SDK Components	282
Upgrading the Android SDK and Tools	284
Debugging with the Amazon Kindle Fire	284
Configuring Other Android Devices for Development Purposes	286

B Eclipse IDE Tips and Tricks 289

Organizing Your Eclipse Workspace	289
Writing Code in Java	293

C Supplementary Materials 299

Using the Source Code for This Book 299

Accessing the Android Developer Website 300

Accessing the Publisher's Website 300

Accessing the Authors' Website 301

Contacting the Authors 303

Leveraging Online Android Resources 304

Index 305

Acknowledgments

This book would never have been written without the guidance and encouragement we received from a number of very patient and supportive people, including our editorial team, coworkers, friends, and family.

Throughout this project, our editorial team at Pearson has been top notch. Special thanks go to Laura Lewin and Trina MacDonald, Olivia Basegio, Sheri Cain, and the rest of the Pearson publishing team. Our technical reviewers, Ray Rischpater, Mike Wallace, and Jim Hathaway, helped us ensure that this book provides accurate information. With each edition, this book gets better. However, it wouldn't be here without the help of many folks on past editions. Thanks go out to past reviewers, technical editors, and readers for their valuable feedback. Finally, we want to thank our friends and family members who supported us when we needed to make our deadlines.

About the Authors

Lauren Darcey is responsible for the technical leadership and direction of a small software company specializing in mobile technologies, including Android, Apple iOS, BlackBerry, Palm Pre, BREW, and J2ME and consulting services. With more than two decades of experience in professional software production, Lauren is a recognized authority in application architecture and the development of commercial-grade mobile applications. Lauren received a B.S. in Computer Science from the University of California, Santa Cruz.

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

Shane Conder has extensive development experience and has focused his attention on mobile and embedded development for the past decade. He has designed and developed many commercial applications for Android, Apple iOS, BREW, BlackBerry, J2ME, Palm, and Windows Mobile—some of which have been installed on millions of phones worldwide. Shane has written extensively about the mobile industry and evaluated mobile-development platforms on his tech blogs and is well known within the blogosphere. Shane received a B.S. in Computer Science from the University of California.

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

The authors have also published several other Android books, including *Android Wireless Application Development*, *Android Wireless Application Development Volume I: Android Essentials*, *Android Wireless Application Development Volume 2: Advanced Topics*, *Sams Teach Yourself Android Application Development*, and the mini-book *Introducing Android Development with Ice Cream Sandwich*. Lauren and Shane have also published numerous articles on mobile-software development for magazines, technical journals, and online publishers of educational content. You can find dozens of samples of their work in *Linux User and Developer*, *Smart Developer* magazine (Linux New Media), *developer.com*, *Network World*, *Envato* (MobileTuts+ and

CodeCanyon), and InformIT, among others. They also publish articles of interest to their readers at their own Android website: <http://androidbook.blogspot.com>. You can find a full list of the authors' publications at <http://goo.gl/f0Vlj>.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book. Please contact the authors for technical help related to this book at androidwirelessdev+kf1@gmail.com.

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the authors and editors who worked on the book.

Email: errata@informit.com

Mail: Addison-Wesley
ATTN: Reader Feedback
1330 Avenue of the Americas
35th Floor
New York, New York, 10019

Reader Services

Visit our website and register this book at <http://informit.com/register> for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

The Amazon Kindle Fire is one of the most exciting and popular new Android devices available to consumers and application developers alike. It's so popular, in fact, that in the six weeks the Amazon Kindle Fire was available for purchase at the end of 2011, 3.9 million units were sold, accounting for over 14 percent of the tablet market, making it the #1-selling Android tablet (<http://goo.gl/f9hFM>). This book covers Android Fundamentals for Kindle Fire application development. Here, you are introduced to Android, become familiar with the Android SDK and tools, install the development tools, write your first Android application, and deploy it to a Kindle Fire device. You are also introduced to the design principles necessary to write Android applications, including how Android applications are designed, developed, and published on the Amazon Appstore.

Target Audience for This Book

There's no reason anyone with an Amazon Kindle Fire device, a good idea for a mobile application, and some programming knowledge couldn't put this book to use for fun and profit. Whether you're a programmer looking to break into mobile technology or an entrepreneur with a cool app idea, this book can help you realize your goals of making killer Android apps for devices like the Kindle Fire.

We make as few assumptions about you, as a reader of this book, as possible. No wireless development experience is necessary. We do assume that you're somewhat comfortable installing applications on a computer (for example, Eclipse, the Java JDK, and the Android SDK) and tools and drivers (for USB access to a phone). We also assume that you own at least one Amazon Kindle Fire device and can navigate your way around it for testing purposes.

Android apps are written in Java. Therefore, we assume you have a reasonably solid understanding of the Java programming language (classes, methods, scoping, object-oriented programming [OOP], and so on), ideally using the Eclipse development environment. Familiarity with common Java packages, such as `java.lang`, `java.net`, and `java.util`, will serve you well.

Android can also be a fantastic platform for learning Java, provided that you have some background in OOP and adequate support, such as a professor or some really good Java programming references. We make every attempt to avoid using any fancy or confusing Java in this book, but you will find that, with Android, certain syntactical Java wizardry not often covered

in your typical beginner's Java book is used frequently: anonymous inner classes, method chaining, templates, reflection, and so on. With patience—and some good Java references—even beginning Java developers should be able to make it through this book alive; those with a solid understanding of Java should be able to take this book and run with it without issue.

Finally, regardless of your specific skill set, we expect you to use this book in conjunction with other supplementary resources, specifically the Android SDK reference and the sample source code that accompanies each coding chapter. The Android SDK reference provides exhaustive documentation about each package, class, and method of the Android SDK. It's searchable online. If we were to duplicate this data in book form, this book would weigh a ton—literally.

How This Book Is Structured

In just a handful of straightforward lessons, you'll design and develop a fully functional Android application specifically for Amazon Kindle Fire devices. Each lesson builds on your knowledge of newly introduced Android concepts, and you'll iteratively improve your application from chapter to chapter. You'll also find numerous exercises at the end of each chapter to help cement your newfound knowledge.

This book is divided into four parts:

- **Part I: Kindle Fire Fundamentals**— Here, you get an introduction to Android, become familiar with the Android SDK and tools, install the development tools, and write your first Android application. Part I also introduces the design principles necessary to write Android applications, including how Android applications are structured and configured, as well as how to incorporate application resources such as strings, graphics, and user-interface components into your projects. You also learn how to start developing for a specific Android device: the Amazon Kindle Fire.
- **Part II: Building an Application Framework**— In this part, you begin developing an application called *Have You Read That?*, which serves as the primary teaching tool for the rest of this book. You start by developing its animated splash screen, followed by screens for the main menu, settings, help, and scores. You learn basic user-interface design principles, how to collect input from the user, and how to display dialogs to the user. You implement the core game logic and leverage the network. Finally, you explore some of the secondary issues of application development, like internationalization.
- **Part III: Publishing Your Kindle Fire Application**— Here, you learn what you need to do to prepare for and publish your Android applications to the Amazon Appstore.
- **Part IV: Appendixes**— Here is where you can find several helpful references for setting up your Android development environment for Amazon Kindle Fire development, using the Eclipse IDE, and accessing supplementary book materials, such as this book's websites and downloadable source code.

What Is (and Isn't) in This Book

First and foremost, this book provides a thorough introduction to the Android platform for Kindle Fire application development. In this, we begin with the fundamentals, try to cover the most important aspects of development, and provide information on where to go for more information. This is not an exhaustive reference on the Android SDK. We assume that you will use this book as a companion to the Android SDK documentation, available for download as part of the SDK and online at <http://developer.android.com>.

We only have a short amount of time to get you, the reader, up to speed on the fundamentals of Android development, so forgive us if we stay strictly to the topic at hand. Therefore, we take the prerequisites listed earlier seriously. This book will not teach you how to program, explain Java syntax and programming techniques, or stray too far into the details of supporting technologies often used by mobile applications, like algorithm design, network protocols, developing web servers, graphic design, database schema design, and other such peripheral topics; there are fantastic references available on each of these subjects.

The Android SDK and related tools are updated frequently (every few months). This means that, no matter how hard we try, some minor changes in step-by-step instructions may occur if you choose to use versions of the tools that do not match those listed in the next section, "Development Environment Used." This book is written for Kindle Fire app developers, so it focuses on the Android SDK version used by this specific device.

Because this book helps developers write applications for the Amazon Kindle Fire, we specifically targeted Android SDK Version 2.3.4 (API Level 10) for the tutorial in this book. Still, we make every effort to make this book compatible with other versions of Android, as well as work smoothly regardless of what version of the Android SDK you want to target.

This book is written in a tutorial style. If you're looking for an exhaustive reference on Android development, with cookbook-style code examples and a more thorough examination of the many features of the Android platform, we recommend our more advanced Android books, *Android Wireless Application Development Volume I: Android Essentials* (Third Edition) (ISBN 978-0321813831) and *Android Wireless Application Development Volume II: Advanced Topics* (Third Edition) (ISBN 978-0321813848), which are part of the Addison-Wesley Developer's Library series.

Development Environment Used

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

- Windows 7 and Mac OS X 10.7.x.
- Eclipse Java IDE Version 3.7 (Indigo).
- Android ADT Plug-in for Eclipse, 16.0.1.

- Android SDK Tools, Release 16.
- Sun Java SE Development Kit (JDK) 6 Update 21.
- Code examples target Android SDK API Level 10.
- The code was tested on the original Android Kindle Fire (Android SDK 2.3.4, API Level 10).
- The network portions of the sample application leverage Google App Engine.

Conventions Used in This Book

This book uses the following code-related conventions:

- Code and programming terms are set in a `monospace` font.
- ➞ signifies that the code that follows should appear on the same line as the preceding code.
- Exception handling and error checking are often removed from printed code samples for clarity and to keep the book a reasonable length.

This book uses the following conventions for step-by-step instructions and explanations:

- The core application developed in this book is developed iteratively. Generally, this means that the first time a new concept is explained, every item related to the new concept is discussed in detail. As we move on to more advanced topics in later lessons, we assume that you have mastered some of the more rudimentary aspects of Android development from previous chapters, and we do not repeat ourselves much. In some cases, we instruct you to implement something in an early lesson and then help you improve it in a later chapter.
- We assume that you'll read the chapters of this book in order. As you progress through this book, you'll note that we do not spell out each and every step that must be taken for each and every feature you implement to follow along in building the core application example. For example, if three buttons must be implemented on a screen, we walk you step-by-step through the implementation of the first button but leave the implementation of the other two buttons as an exercise for you. In a later chapter on a different topic, we might simply ask you to implement some buttons on another screen.
- Where we tell you to navigate through menu options, we separate options using commas. For example, if we told you to open a new document, we'd say "Select File, New Document."

About the Short Links

We chose to make many Internet links in this book short links. This benefits the readers of the print book by making it far easier and less prone to error when typing links. These links are all shortened with the `goo.gl` link shortener, which is a service provided by Google. If the target of the link goes away, neither the original link nor the shortened link would work. We're confident that this is the easiest way for readers to effectively use the links we provide. In addition, as authors, we get to see which links readers are actually using.

Sometimes, link shorteners are used as a way to hide nefarious links. Be assured that we have only shortened links that we believe to be valid and safe. In addition, Google provides screening of the target URLs for malware, phishing, and spam sites. Should a target link change hands and become a bad link, using the shortened link will provide you, the reader, with an extra layer of protection.

For more information on this subject, see <http://www.google.com/support/websearch/bin/answer.py?answer=190768> (<http://goo.gl/iv8c7>).

Code Examples for This Book

The source code is available for download on the publisher's website (www.informit.com/title/9780321833976) and on the authors' website (<http://androidbook.blogspot.com/p/book-code-downloads.html>).

We provide complete, functional code projects for each coding chapter in this book. If you're having trouble building the tutorial application as you go along, compare your work to the sample code for that chapter. The sample code is not intended to be the "answer," but it is the complete code listings that could not otherwise be reproduced in a book of this length.

Supplementary Tools Available

Shane Conder and Lauren Darcey, the authors, also run a blog at <http://androidbook.blogspot.com>, where you can always download the latest source code for their books. This website also covers a variety of Android topics and reader discussions, questions, clarifications, the occasional exercise walkthrough, and lots of other information about Android development. You can also find links to their various technical articles online and in print.

Contacting the Authors

Feel free to contact us if you have specific questions; we often post addendum information or tool-change information on our book website:

<http://androidbook.blogspot.com>

You can also email us at androidwirelessdev+kf1@gmail.com.

This page intentionally left blank



Kindle Fire Fundamentals

- 1** Getting Started with Kindle Fire 9
- 2** Mastering the Android Development Tools 29
- 3** Building Kindle Fire Applications 39
- 4** Managing Application Resources 53
- 5** Configuring the Android Manifest File 69
- 6** Designing an Application Framework 83

This page intentionally left blank

Getting Started with Kindle Fire

Android is the first *complete, open, and free* mobile platform. Developers enjoy a comprehensive software development kit (SDK), with ample tools for developing powerful, feature-rich applications. The platform is open source, relying on tried-and-true open standards with which developers will be familiar. Best of all, there are no costly barriers to entry for developers: no required fees. (A modest fee is required to publish on third-party distribution mechanisms, such as the Android Market.) Android developers have numerous options for distributing and commercializing their applications.

Introducing Android

To understand where Android fits with other mobile technologies, let's take a minute to talk about how and why this platform came about.

Google and the Open Handset Alliance

In 2007, a group of handset manufacturers, wireless carriers, and software developers (notably, Google) formed the Open Handset Alliance, with the goal of developing the next generation of wireless platform. Unlike existing platforms, this new platform would be nonproprietary and based on open standards, which would lead to lower development costs and increased profits. Mobile software developers would also have unprecedented access to the handset features, allowing for greater innovation.

As proprietary platforms, such as RIM BlackBerry and Apple iPhone, gained traction, the mobile development community eagerly listened for news of this potential game-changing platform.

Android Makes Its Entrance

In 2007, the Open Handset Alliance announced the Android platform and launched a beta program for developers. Android went through the typical revisions of a new platform. Several preview versions of the Android SDK were released. The first Android handset (the T-Mobile G1) began shipping in late 2008. Throughout 2009 and 2010, new and exciting Android smartphones reached markets throughout the world, and the platform proved itself to industry and consumers alike. Over the last three years, numerous revisions to the Android platform have been rolled out, each providing compelling features for developers to leverage and users to enjoy. Recently, mobile platforms began to consider devices above and beyond the traditional smartphone paradigm to other devices, such as tablets, ebook readers, and set-top boxes, like Google TV.

As of this writing, hundreds of Android devices are available to consumers around the world—from high-end smartphones to low-end “free with contract” handsets and everything in between. This figure does not include the numerous Android tablet and e-book readers also available, the dozens of upcoming devices already announced, or the consumer electronics running Android. (For a nice list of Android devices, check out this Wikipedia link: <http://goo.gl/fU2X5>.) More than 450,000 applications are currently published on the Android Market (now called Google Play), and there are more than 30,000 applications on the Amazon Appstore for Android. In the United States, all major carriers now carry Android phones prominently in their product lines, as do many in Asia, Europe, Central/South America, and beyond. The rate of new Android devices reaching the world markets continues to increase.

Google has been a contributing member of the Open Handset Alliance from the beginning. The company hosts the Android open source project and the developer website (<http://developer.android.com>). This website is your go-to site for downloading the Android SDK, getting the latest platform documentation, and browsing the Android developer forums. Google also runs the most popular service for selling Android applications to end users: the Android Market. The Android mascot is a little green robot (see Figure 1.1).

Cheap and Easy Development

If there's one time when “cheap and easy” is a benefit, it's with mobile development. Wireless application development, with its ridiculously expensive compilers and preferential developer programs, has been notoriously expensive to break into compared to desktop development. Here, Android breaks the proprietary mold. Unlike other mobile platforms, there are virtually no costs to developing Android applications.

The Android SDK and tools are freely available on the Android developer website (<http://developer.android.com> [<http://goo.gl/K8GgDJ>]). The freely available Eclipse program has become the most popular integrated development environment (IDE) for Android application development; there is a powerful plug-in available on the Android developer site for facilitating Android development with Eclipse.



Figure 1.1 The Android Mascot (The Bugdroid)

So, we covered cheap; now let's talk about why Android development is easy. Android applications are written in Java, which is one of the most popular development languages around. Java developers will be familiar with many of the packages provided as part of the Android SDK, such as `java.net`. Experienced Java developers will be pleased to find that the learning curve for Android is reasonable.

In this book, we focus on the most common, popular, and simple setup for developing Android applications:

- We use the most common and supported development language: Java. Although we do not teach you Java, we try our best to keep the Java code we use simple and straightforward so that even beginners won't wrestle with syntax. Even so, if you are new to Java, we recommend *Sams Teach Yourself Java in 24 Hours* by Rogers Cadenhead and *Thinking in Java* by Bruce Eckel, Fourth Edition in Print (Third Edition free from <http://goo.gl/mtjoz>—a zip file from Bruce Eckel's site at <http://www.mindviewinc.com/Books/>).
- We use the most popular development environment: Eclipse. It's free, it's well supported by the Android team, and it's the only supported IDE compatible with the Android Development Tools plug-in. Did we mention it's free?

- We write instructions for the most common operating system used by developers: Windows. Users of Linux or Mac may need to translate some keyboard commands, paths, and installation procedures.
- We focus on the Android platform version available on the Amazon Kindle Fire: Android 2.3.4 (API Level 10).

If you haven't installed the development tools needed to develop Android applications or the Android SDK and tools yet, do so at this time.

Let's get started!

Note

You can find all the details of how to install and configure your computer for Android application development in Appendix A, "Configuring Your Android Development Environment." You will need to install and configure Java, Eclipse, the Android SDK, and the ADT plug-in for Eclipse. You will need to configure ADB and possibly USB drivers for connecting your development machine to Kindle Fire for debugging. Again, all this is covered in Appendix A.

Familiarizing Yourself with Eclipse


Let's begin by writing a simple Android "Hello, World" application that displays a line of text to the user. As you do so, you also tour the Eclipse environment. Specifically, you learn about some of the features offered by the Android development tools (ADT) plug-in for Eclipse. The ADT plug-in provides functionality for developing, compiling, packaging, and deploying Android applications. Specifically, the ADT plug-in provides the following features:

- The Android project wizard, which generates all the required project files
- Android-specific resource editors, including a graphical layout editor for designing Android application user interfaces
- The Android SDK and the AVD Manager
- The Eclipse DDMS perspective for monitoring and debugging Android applications
- Integration with the Android LogCat logging utility
- Integration with the Android Hierarchy Viewer layout utility
- Automated builds and application deployment to Android emulators and devices
- Application packaging and code signing tools for release deployment, including ProGuard support for code optimization and obfuscation

Now, let's take some of these features for a spin.

Creating Android Projects

The Android project wizard creates all the required files for an Android application. Open Eclipse and follow these steps to create a new project:

1. Choose File, New, Android Project or click the Android Project creator icon  on the Eclipse toolbar.

Note

The first time you try to create an Android Project in Eclipse, you might need to choose File, New, Project, and then select Android, Android Project. After you do this once, the Android project type appears in the Eclipse project types, and you can use the method described in step 1.

2. Choose a project name. In this case, name the project `HelloKindle`.
3. Choose a location for the project source code. Because this is a new project, select the Create New Project in Workspace radio button. If you prefer to store your project files in a location other than the default, simply uncheck the Use Default Location checkbox and browse to the directory of your choice. The settings should look like Figure 1.2.

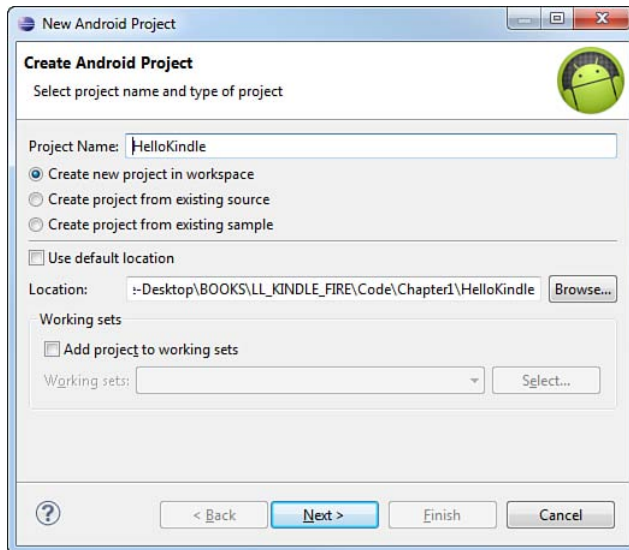


Figure 1.2 Project Name and Location

4. Click the Next button.
5. Select a build target for your application, as shown in Figure 1.3. For most applications, you want to select the version of Android that's most appropriate for the devices used by your target audience and the needs of your application. For Kindle development, choose API Level 10 (Android 2.3.3) using the Android Open Source Project vender version (not the Google, Inc., vender version). Kindle Fire devices do not have access to Google add-ons.

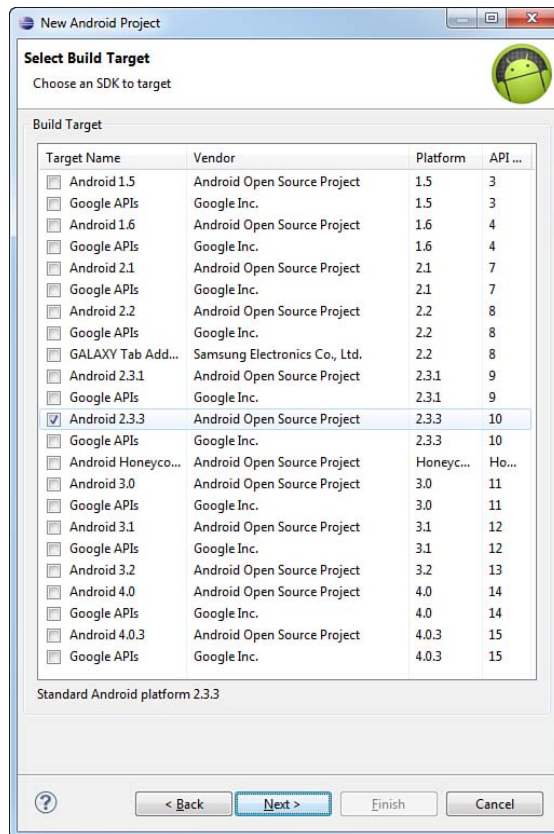



Figure 1.3 Choose SDK Target

6. Click the Next button.
7. Specify an application name. This name is what users will see. In this case, call the application `Hello Kindle`.

8. Specify a package name, following standard package-namespace conventions for Java. Because all code in this book falls under the `com.kindlebook.*` namespace, use the package name `com.kindlebook.hellokindle`.
9. If needed, check the Create Activity checkbox. This instructs the wizard to create a default launch `Activity` class for the application. Call your activity `HelloKindleActivity`.
10. Confirm that the Minimum SDK field is correct. This field will be set to the API level of the build target by default. (Android 2.3.3 is API Level 10.) If you want to support older versions of the Android SDK, you need to change this value. For example, to support devices with Android 1.6, set the Minimum SDK field to API Level 4. The Kindle is based on API Level 10, however, so an application just targeting the Kindle does not need to worry about this. Your project settings will look like what's shown in Figure 1.4.
11. The Android project wizard allows you to create a test project in conjunction with your Android application, also shown in Figure 1.4. For this example, a test project is unnecessary. However, you can always add a test project later by clicking the Android Test Project creator icon, which is to the right of the Android project wizard icon  on the Eclipse toolbar. Test projects are discussed further in Chapter 16, "Testing Kindle Fire Applications."
12. Click the Finish button.

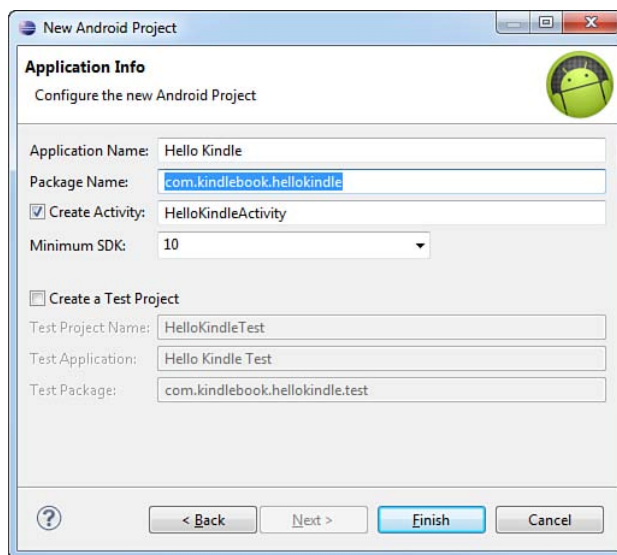


Figure 1.4 Configure Package Name, Initial Activity, and Minimum SDK

Note

You can also add existing Android projects to Eclipse by using the Android project wizard. To do this, simply select Create Project from Existing Source instead of the default Create New Project in Workspace in the New Android Project dialog (refer to Figure 1.2). Several sample projects are provided in the `/samples` directory of the Android SDK, under the specific platform they support. For example, the Android SDK sample projects are found in the directory `/platforms/android-xxx/samples` (where `xxx` is the platform level number, such as 10).

You can also select a third option: Create Project from Existing Sample, which will do as it says. However, make sure that you choose the build target first option to get the list of sample projects that you can create.

Exploring Your Android Project Files

You will now see a new Android project called `HelloKindle` in the Eclipse File Explorer. In addition to linking the appropriate Android SDK jar file, the following core files and directories are created:

- `AndroidManifest.xml`—The central configuration file for the application.
- `project.properties`—A generated build file used by Eclipse and the Android ADT plug-in. Do not edit this file.
- `proguard.cfg`—A generated build file used by Eclipse, ProGuard, and the Android ADT plug-in. Edit this file to configure your code optimization and obfuscation settings for release builds.
- `/src` **folder**—Required folder for all source code.
- `/src/com.kindlebook.hellokindle/HelloKindleActivity.java`—Main entry point to this application, named `HelloKindleActivity`. This activity has been defined as the default launch activity in the Android manifest file.
- `/gen/com.kindlebook.hellokindle/R.java`—A generated resource management source file. Do not edit this file.
- `/assets` **folder**—Required folder where uncompiled file resources can be included in the project.
- `/res` **folder**—Required folder where all application resources are managed. Application resources include animations, drawable graphics, layout files, data-like strings and numbers, and raw files.
- `/res/drawable-*` **folders**—Application icon graphic resources are included in several sizes for different device screen resolutions.
- `/res/layout/main.xml`—Layout resource file used by `HelloKindleActivity` to organize controls on the main application screen.
- `/res/values/strings.xml`—The resource file where string resources are defined.

Editing Project Resources

The Android manifest file is the central configuration file for an Android application. Double-click the `AndroidManifest.xml` file within your new project to launch the Android manifest file editor (see Figure 1.5).

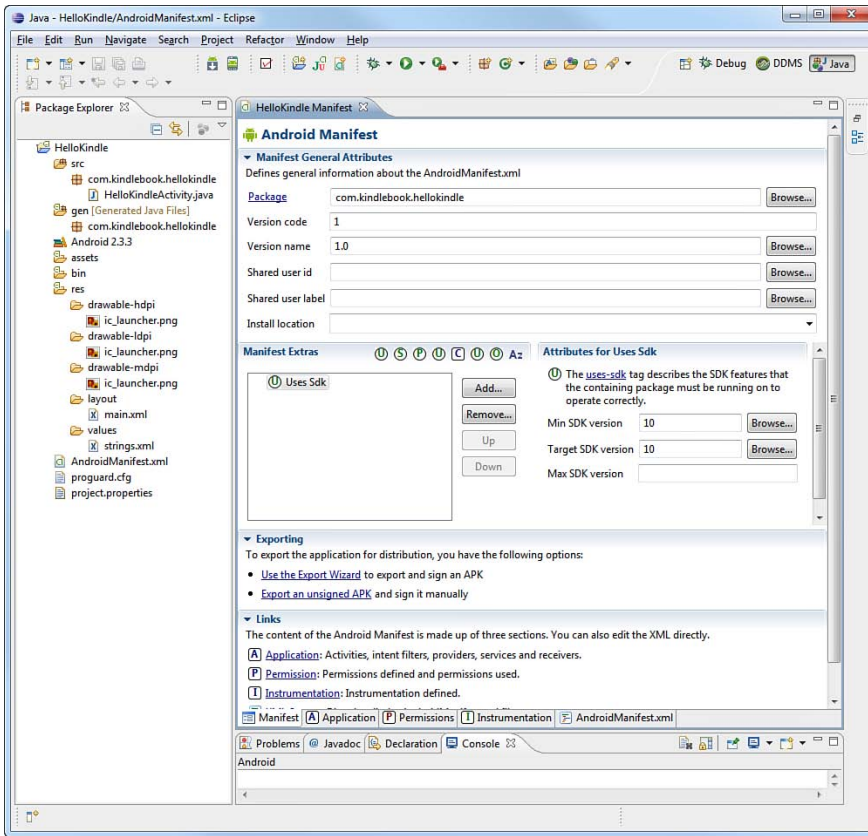



Figure 1.5 Editing the Android Manifest File in Eclipse

Editing the Android Manifest File

The Android manifest file editor organizes the manifest information into numerous tabs:

- **Manifest**—This tab, shown in Figure 1.5, is used for general application-wide settings, such as the package name and application version information (used for installation and upgrade purposes).

- **Application**—This tab is used to define application details, such as the name and icon the application displays, as well as the “guts” of the application, such as what activities can be run (including the default launch `HelloKindleActivity`) and other functionality and services that the application provides.
- **Permissions**—This tab is used to define the application’s permissions. For example, if the application requires the ability to access Internet resources, it must register a `uses-permission` tag within the manifest, with the name `android.permission.INTERNET`.
- **Instrumentation**—This tab is used for unit testing, using the various instrumentation classes available within the Android SDK.
- **AndroidManifest.xml**—This tab provides a simple XML editor to directly edit the manifest file. Because all Android resource files, including the Android manifest file, are simply XML files, you can always edit the XML instead of using the resource editors. You can create a new Android XML resource file by clicking the Android XML creator icon  on the Eclipse toolbar.

If you switch to the `AndroidManifest.xml` tab, your manifest file will look something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.kindlebook.hellokindle"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="10" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".HelloKindleActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action
                    android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Editing Other Resource Files

Android applications are made up of functions (Java code, classes) and data (including resources like graphics, strings, and so on). Most Android application resources are stored under the `/res`

subdirectory of the project. The following subdirectories are also available by default in a new Android project:

- `/drawable-ldpi`, `/drawable-hdpi`, `/drawable-mdpi`—These subdirectories store graphics and drawable resource files for different screen densities and resolutions. If you browse through these directories using the Eclipse Project Explorer, you will find the `icon.png` graphics file in each one; this is your application's icon.
- `/layout`—This subdirectory stores user interface layout files. Within this subdirectory, you will find the `main.xml` screen layout resource file, which defines the user interface for the one activity in this simple application.
- `/values`—This subdirectory organizes the various types of resources, such as text strings, color values, and other primitive types. Here, you find the `strings.xml` resource file, which contains all the string resources used by the application.

If you double-click any of the resource files, the resource editor launches. Remember that you can always directly edit the XML. For example, let's try editing a string resource file. If you inspect the `main.xml` layout file of the project, you notice that it displays a simple layout with a single `TextView` control. This user-interface control simply displays a string. In this case, the string displayed is defined in the string resource called `@string/hello`. To edit the string resource called `@string/hello` using the string resource editor, follow these steps:

1. Open the `strings.xml` file in the resource editor by double-clicking it in the Package Explorer of Eclipse.
2. Select the `String` called `hello` and note the name (`hello`) and value (`Hello World, HelloKindleActivity!`) shown in the resource editor.
3. Within the Value field, change the text to `Hello, Kindle Fire`.
4. Save the file.

If you switch to the `strings.xml` tab and look through the raw XML, you notice that two string elements are defined within a `<resources>` block:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Kindle Fire</string>
    <string name="app_name">Hello Kindle</string>
</resources>
```

The first resource is the string called `@string/hello`. The second resource is the string called `@string/app_name`, which contains the name label for the application. If you look at the Android manifest file again, you see `@string/app_name` used in the application configuration.

We talk more about project resources in Chapter 4, "Managing Application Resources." For now, let's move on to compiling and running the application.

Running and Debugging Applications


To build and debug an Android application, you must first configure your project for debugging. The ADT plug-in enables you to do this entirely within the Eclipse development environment. Specifically, you need to do the following:

- Create and configure an Android Virtual Device (AVD).
- Create an Eclipse debug configuration for your project.
- Build the Android project and launch the emulator with the new AVD.

When you complete each of these tasks, Eclipse attaches its debugger to the Android emulator (or Android device connected via USB), and you are free to run and debug the application as desired.

Managing Android Virtual Devices

To run an application in the Android emulator, you must configure an AVD. The AVD profile describes the type of device you want the emulator to simulate, including which Android platform to support. You can specify different screen sizes and resolutions, and you can specify whether the emulator has an SD card and, if so, its capacity. In this case, a slightly modified AVD for the default installation of Android 2.3.3 will suffice. Here are the steps for creating a basic AVD:

1. Launch the Android Virtual Device Manager from within Eclipse by clicking the little Android icon with the bugdroid in mini-phone  on the toolbar. You can also launch the manager by selecting Window, AVD Manager in Eclipse.
2. Click the New button to create a new AVD.
3. Choose a name for the AVD. Because you are going to take all the defaults, name this AVD `KindleFire-Portrait`.
4. Choose a build target. The Kindle Fire is based on API Level 10 (Android 2.3.3). Remember not to use a Google API version, because Kindle Fire does not support this.
5. Choose an SD card capacity, in either kibibytes or mibibytes. (Not familiar with kibibytes? See this Wikipedia entry: <http://goo.gl/N3Rdd>.)

Note

Although to mimic a Kindle Fire, you'd choose 8GiB, we recommend choosing something fairly small, because a file of the size of the SD card will be allocated on your drive each time you create a new AVD; these can add up quickly. Unless your application requires substantial storage, we recommend something like 64MiB.

6. Choose a skin. This option controls the different visual looks of the emulator. In this case, we use the effective resolution of the Kindle Fire screen of 600 pixels wide and

1004 pixels high (the default portrait resolution). Alternatively, we could create an AVD for landscape mode, where we'd need to use 1024-pixels wide and 580-pixels high. The Kindle Fire reserves some space for a soft key menu.

7. Under Hardware, change the Abstracted LCD Density to 169 and change the Device RAM Size to 512 to better emulate the Kindle Fire device characteristics.
8. Optionally, enable the Snapshot feature. This allows you to save and restore the state of an emulator session, which dramatically improves the speed with which it launches.

Your project settings should look like what's shown in Figure 1.6.

9. Click the Create AVD button and wait for the operation to complete. This may take a few seconds if your SD card capacity is large, because the memory allocated for the SD card emulation is formatted as part of the AVD creation process. You should now see your newly created AVD in the list.

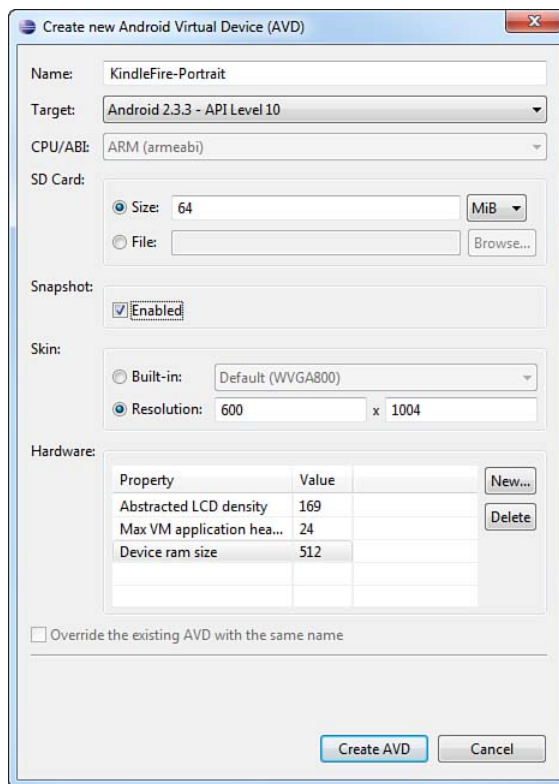



Figure 1.6 Creating a New AVD in Eclipse

Creating Debug and Run Configurations in Eclipse

You are almost ready to launch your application. You have one last task remaining: You need to create a debug configuration (or a run configuration) for your project in Eclipse. To do this, follow these steps:

1. In Eclipse, choose Run, Debug Configurations from the menu or, alternatively, click the dropdown menu next to the debug icon  on the Eclipse toolbar and choose the Debug Configurations option.
2. Double-click the Android Application item to create a new entry.
3. Edit that new entry, currently called `New_configuration`, by clicking it in the left pane.
4. Change the name of the configuration to `HelloKindleDebug`.
5. Set the project by clicking the Browse button and choosing the `HelloKindle` project.
6. On the Target tab, check the box next to the AVD you created.
7. Apply your changes by clicking the Apply button. Your Debug Configurations dialog should look like what's shown in Figure 1.7.

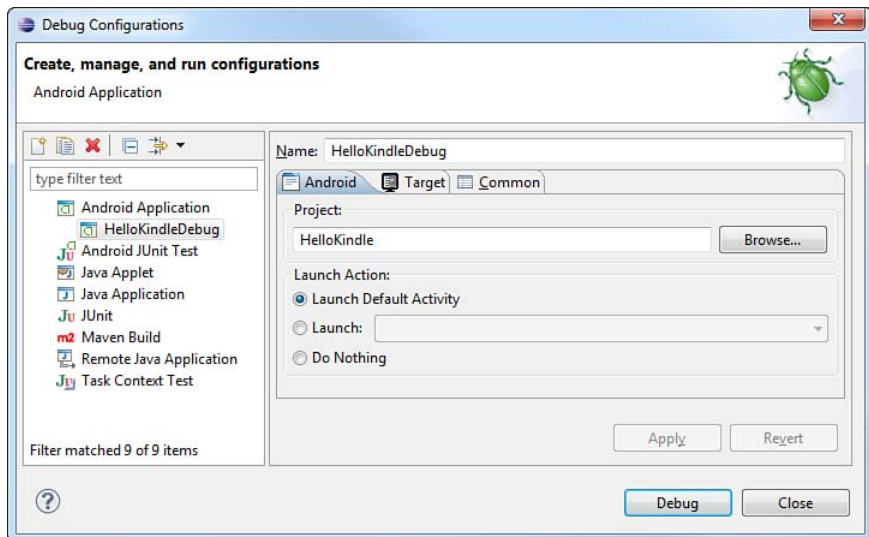



Figure 1.7 The HelloKindleDebug Debug Configuration in Eclipse

Launching Android Applications Using the Emulator

It's launch time, and your application is ready to go! To launch the application, you can simply click the Debug button from within the Launch Configuration screen, or you can do

it from the project by clicking the little green bug icon  on the Eclipse toolbar. Then, select `HelloKindleDebug` debug configuration from the list.

Note

The first time you try to select `HelloKindleDebug` debug configuration from the little green bug dropdown, you have to navigate through the Debug Configuration Manager. Future attempts will show the `HelloKindleDebug` configuration for convenient access.

After you click the Debug button, the emulator launches (see Figure 1.8). This can take some time, so be patient.

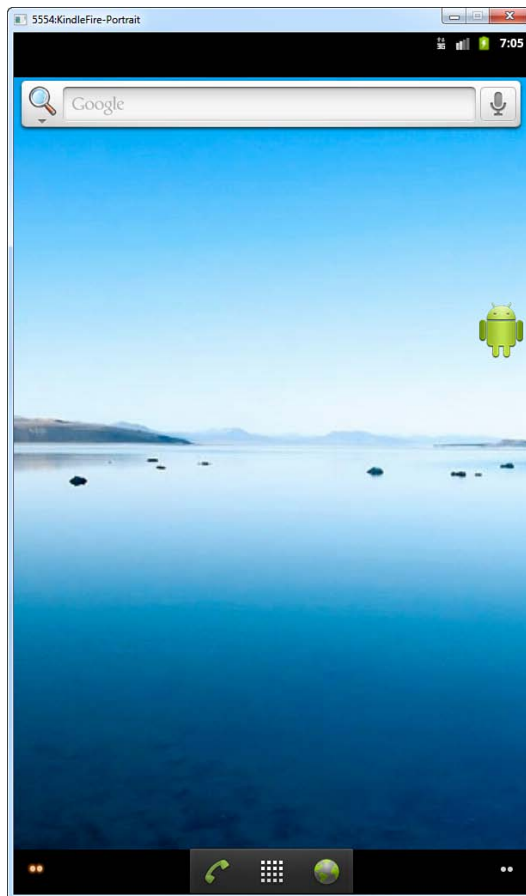


Figure 1.8 The Android Emulator Home Screen

Now, the Eclipse debugger is attached, and your application runs, as shown in Figure 1.9.

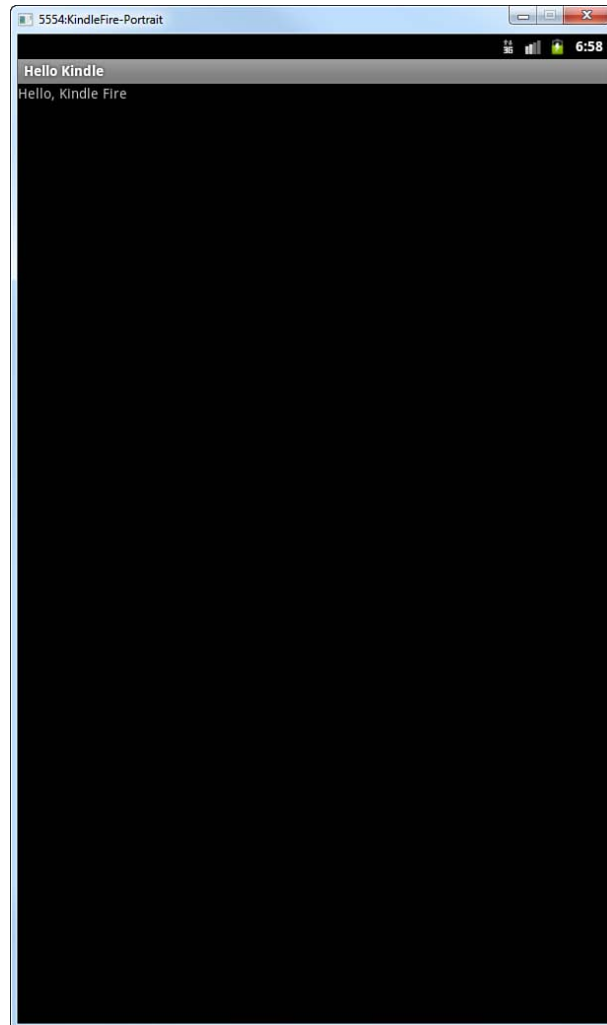


Figure 1.9 The Application Running

As you can see, the application is simple. It displays a single `TextView` control with a line of text. The application does nothing else.


The emulator's home screen doesn't look anything like the home screen on a real Kindle Fire device, because it has been redesigned by Amazon. Among other things, this means that the

emulator won't work for full application testing. You need to get a real Kindle Fire device for that.

Controlling the Emulator

When you create a custom AVD in this way, it will not have the keyboard and control buttons to the left of the screen, like you might be used to with the default emulators. All the commands are available through your development machine keyboard. For example, the Home key maps conveniently to the Home key. The menu key maps to F2 or page-up. Search maps to F5. Back maps to Esc. There are many more; find them in the Android documentation at <http://goo.gl/5DMil>.

Debugging Android Applications Using DDMS

In addition to the normal Debug perspective built into Eclipse for stepping through code and debugging, the ADT plug-in adds the DDMS perspective. While you have the application running, quickly look at this perspective in Eclipse. You can get to the DDMS perspective (see Figure 1.10) by clicking the Android DDMS icon  in the top-right corner of Eclipse. To switch back to the Eclipse Project Explorer, simply choose the Java perspective from the top-right corner of Eclipse.

The DDMS perspective can be used to monitor application processes, as well as interact with the emulator. You can simulate voice calls and send SMS messages to the emulator. You can send a mock location fix to the emulator to mimic location-based services. You learn more about DDMS and the other tools available to Android developers in Chapter 2, “Mastering the Android Development Tools.”

The LogCat logging tool is displayed on both the DDMS perspective and the Debug perspective. This tool displays logging information from the emulator or the device, if a device is plugged in via USB.

Launching Android Applications on a Device

It's time to load your application onto a real Kindle Fire device. To do this, you need to connect the Kindle Fire to your computer using a USB data cable. Make sure that you have your machine configured for Kindle Fire debugging, as discussed in Appendix A.

To ensure that you debug using the correct settings, follow these steps:

1. In Eclipse, from the Java perspective (as opposed to the DDMS perspective), choose Run, Debug Configurations.
2. Single-click HelloKindleDebug Debug Configuration.
3. On the Target tab, change Deployment Target Selection Mode to Manual. You can always change it back to Automatic later, but choosing Manual forces you to choose whether to debug within the emulator (with a specific AVD) or a device, if one is plugged in via USB, whenever you choose to deploy and debug your application from Eclipse.

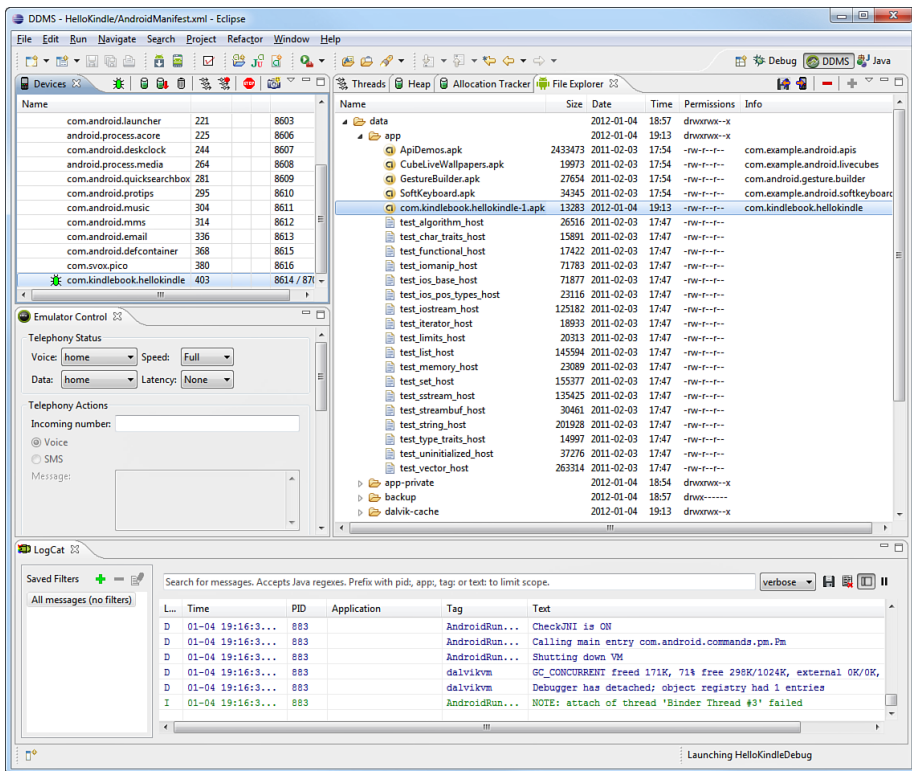


Figure 1.10 The DDMS Perspective

4. Apply your changes by clicking the Apply button.
5. Plug a Kindle Fire device into your development computer by using a USB cable.
6. Click the Debug button within Eclipse. A dialog appears (see Figure 1.11), showing all available configurations for running and debugging your application. All physical devices are listed, as are existing emulators that are running. You can also launch new emulator instances by using other AVDs that you have created.
7. Choose the available Kindle Fire device. If you do not see the Kindle Fire listed, check your cables and make sure that you installed the appropriate drivers, as explained in Appendix A.

Eclipse now installs the Android application onto your Kindle Fire, attaches the debugger, and runs your application. Your device will show a screen similar to the one you saw in the emulator. If you look at the DDMS perspective in Eclipse, you see that logging information is available, and many features of the DDMS perspective work with physical devices and the emulator, such as taking a screenshot (see Figure 1.12).

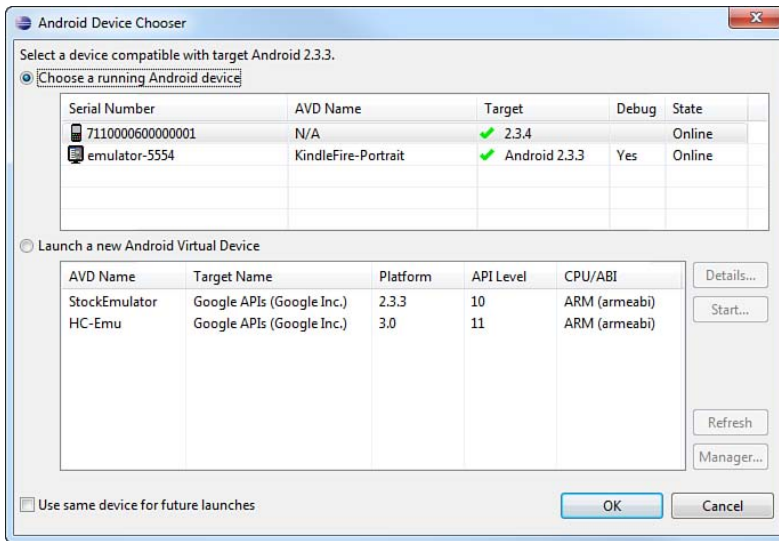


Figure 1.11 Choosing an Application Deployment Target

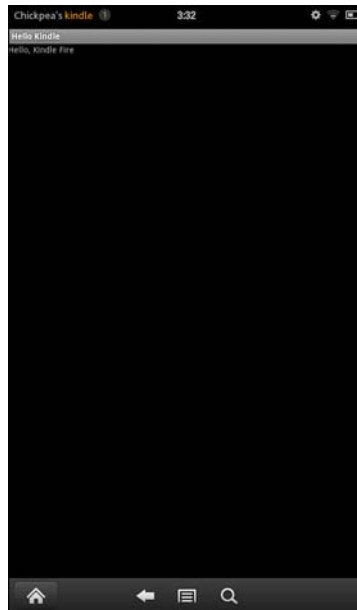


Figure 1.12 The Application Running on a Kindle Fire

New to Eclipse?

If you're still learning the ropes of the Eclipse development environment, now is a great time to check out Appendix B, "Eclipse IDE Tips and Tricks."

Summary

Congratulations! You are now a Kindle Fire Android developer. You have begun to learn your way around the Eclipse development environment. You created your first Android project. You reviewed and compiled working Android code. Finally, you ran your newly created Android application on the Android emulator and on a real Kindle Fire.

Exercises

1. Visit the Android website at <http://developer.android.com> and look around. Check out the online Developer's Guide and reference materials. Check out the Community tab and seriously consider signing up for the Android Beginners and Android Developers Google Groups.
2. Visit the Eclipse website and look around. Check out the online documentation at <http://www.eclipse.org/documentation/> (<http://goo.gl/fc406>). Eclipse is an open source project made freely available; check out the Contribute link (<http://www.eclipse.org/contribute/>) and consider how you might give back to this great project in some way—either by reporting bugs or doing one of the many other options suggested.
3. Visit the Amazon Appstore Developer portal and look around. You can get started here: <https://developer.amazon.com/welcome.html>. While you're at it, head over to Amazon Appstore Developer Blog at <http://www.amazonappstoredev.com>.

Mastering the Android Development Tools

Android developers are fortunate to have more than a dozen development tools at their disposal to help facilitate the design of quality applications. Understanding what tools are available and what they can be used for is a task best done early in the Android learning process, so that when you are faced with a problem, you have some clue as to which utility might be able to help you find a solution. Most of the tools for Android development are integrated into Eclipse using the ADT plug-in, but they can also be launched independently—you'll find the executables in the `/tools` subdirectory of the Android SDK installation. In this chapter, we walk through a number of the most important tools available for use with Android. This information will help you develop Android applications faster and with fewer roadblocks.

Using the Android Documentation

Although it is not a tool per se, the Android documentation is a key resource for Android developers. An HTML version of the Android documentation is provided in the `/docs` subfolder of the Android SDK documentation, and this should always be your first stop when you encounter a problem. You can also access the latest help documentation online at the Android Developer website (<http://developer.android.com> [<http://goo.gl/K8GgDJ>]). (See Figure 2.1 for a screenshot of the Dev Guide tab of this website.)

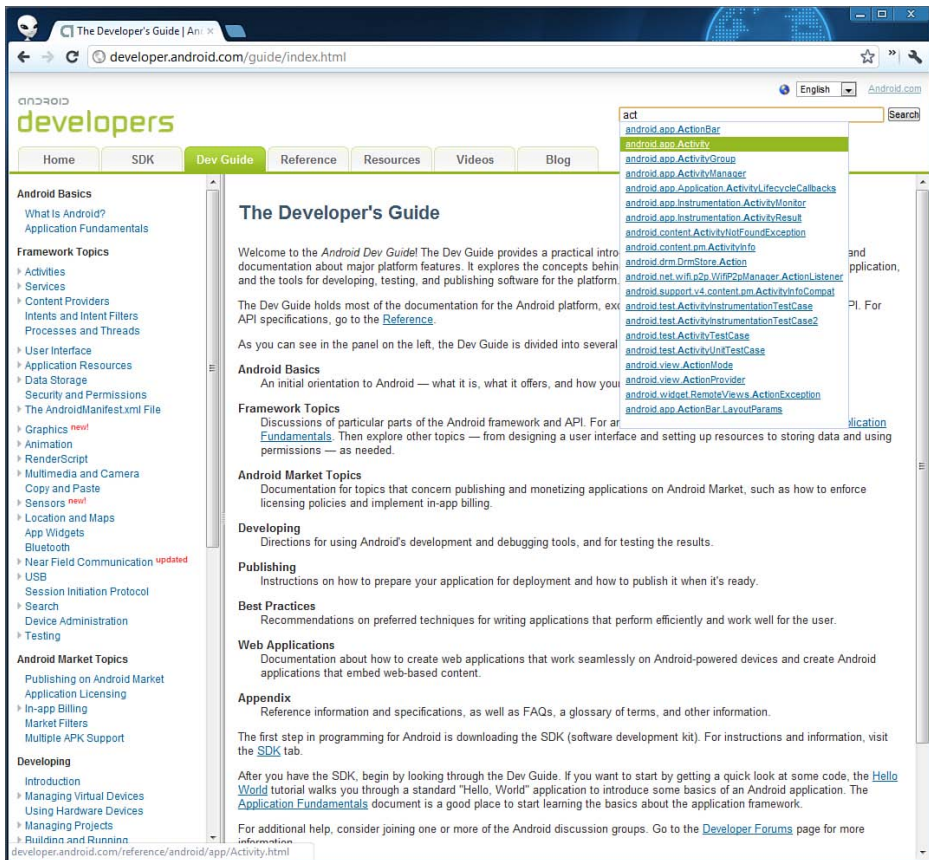


Figure 2.1 The Android Developer Website

The Android documentation is divided into seven sections:

- **Home**—This tab provides some high-level news items for Android developers, including announcements of new platform versions. You can also find quick links for downloading the latest Android SDK, publishing your applications on the Android market, and other helpful information.
- **SDK**—This tab provides important information about the SDK version installed on your machine. One of the most important features of this tab is the release notes, which describe any known issues for the specific installation. This information is also useful if the online help has been upgraded, but you want to develop to an older version of the SDK.

- **Dev Guide**—This tab links to the Android Developer’s Guide, which includes a number of FAQs for developers, best practice guides, and a useful glossary of Android terminology for those new to the platform. The section also lists all Android platform versions (API levels), supported media formats, and lists of intents.
- **Reference**—This tab includes a searchable package and class index of all Android APIs provided as part of the Android SDK (in a Javadoc-style format).
- **Resources**—This tab includes links to articles, tutorials, and sample code, as well as acting as a gateway to the Android developer forums. There are a number of Google groups you can join, depending on your interests.
- **Videos**—This tab, which is available online only, is your resource for Android training videos. Here, you can find videos about the Android platform, developer tips, and the Google I/O conference sessions.
- **Blog**—This tab links to the official Android developer blog. Check here for the latest news and announcements about the Android platform. This is a great place to find how-to examples, learn how to optimize Android applications, and hear about new SDK releases and Android developer challenges.

Now is a good time to get to know your way around the Android SDK documentation. First, check out the online documentation, and then try the local documentation (available in the `/docs` subdirectory of your Android SDK installation).

Debugging Applications with DDMS

The Dalvik Debug Monitor Service (DDMS) is a debugging utility that is integrated into Eclipse through a special Eclipse perspective. The DDMS perspective provides a number of useful features for interacting with emulators or devices and the applications being debugged (see Figure 2.2).

The features of DDMS are roughly divided into six functional areas:

- Task management
- File management
- Memory management
- Emulator interaction
- Logging
- Screen captures

DDMS and the DDMS perspective are essential debugging tools. Now, let’s look at how to use these features in more detail.

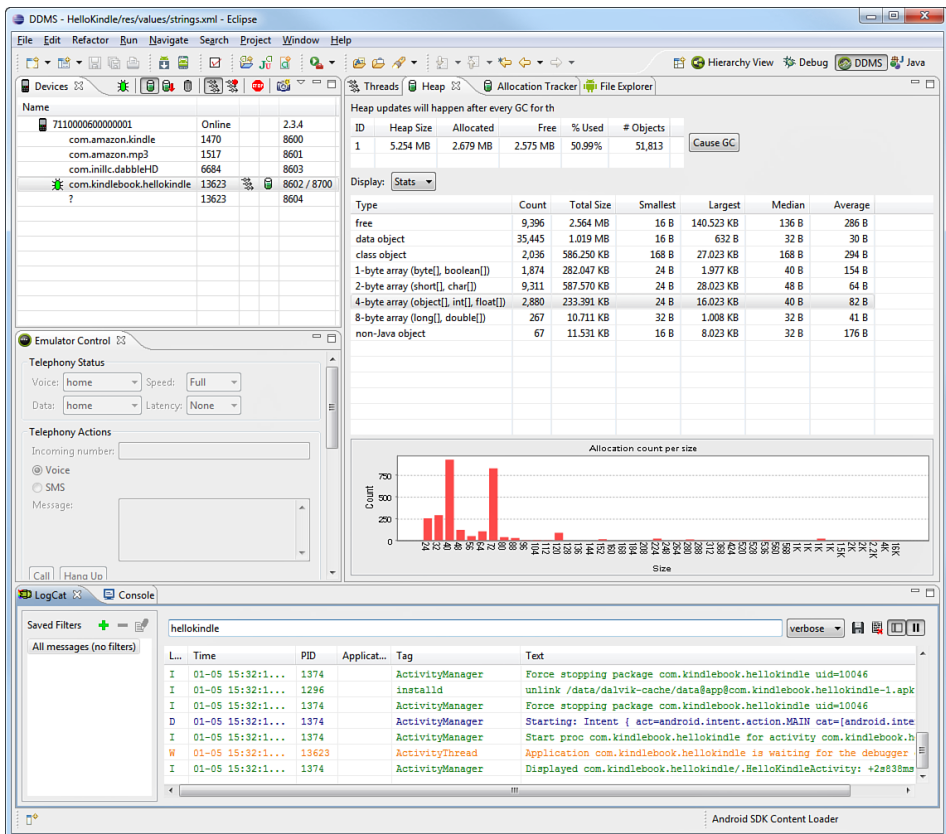






Figure 2.2 The DDMS Perspective in Eclipse

Debugging from the DDMS Perspective

Within the DDMS perspective, you can choose a specific process on an emulator or a device and then click the Debug button  to attach a debugger to that process. You need to have the source code in your Eclipse workspace for this to work properly. This works only in Eclipse, not in the standalone version of DDMS. This is useful if you run into a situation to debug when your application is already running. It can also speed up debugging by launching the application normally and only attaching the debugger when needed.

Managing Tasks

The top-left corner of the DDMS perspective lists the emulators and devices currently connected. You can select individual instances and view its processes and threads. You can inspect threads by clicking the device process you are interested in—for example,

com.androidbook.hellokindle—and clicking the Update Threads button , as shown in Figure 2.3. You can also prompt garbage collection on a process and then view the heap updates by clicking the Update Heap button . Finally, you can stop a process by clicking the Stop Process button .

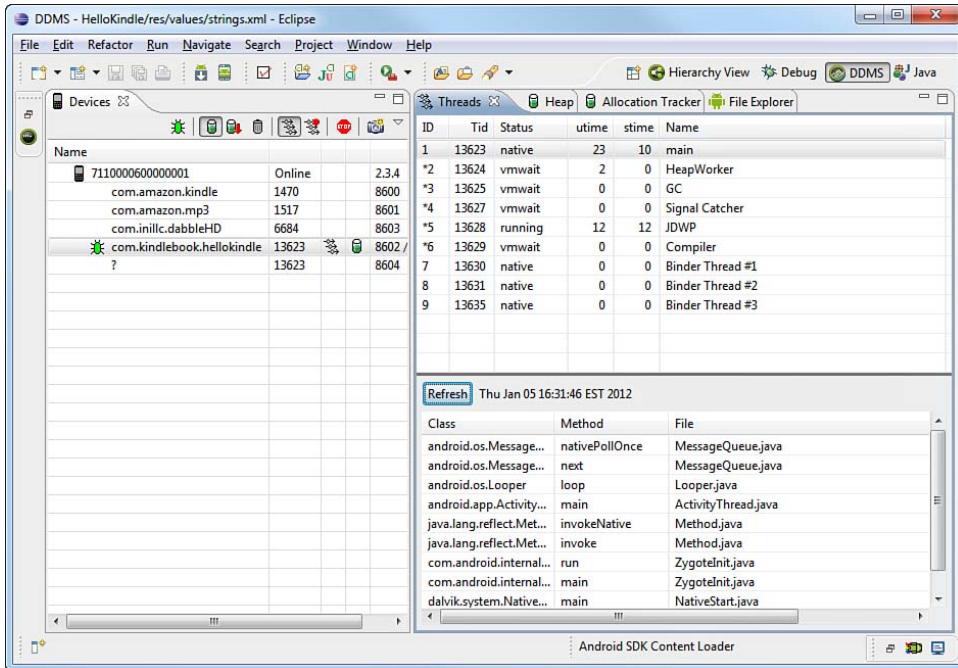





Figure 2.3 The Threads Tab in DDMS

Browsing the Android File System

You can use the DDMS File Explorer to browse files and directories on the emulator or a device (see Figure 2.4). You can copy files between the Android file system and your development machine by using the Push  and Pull  buttons, which are available in the top right-hand corner of the File Explorer tab.

You can also delete files and directories by using the Delete button  or just pressing Delete. There is no confirmation for this delete operation—nor can it be undone.

Note

When using the File Explorer on regular devices, few files and directories will be visible because you do not normally have root access to see these. On the emulator and rooted devices, more of the files will be visible.

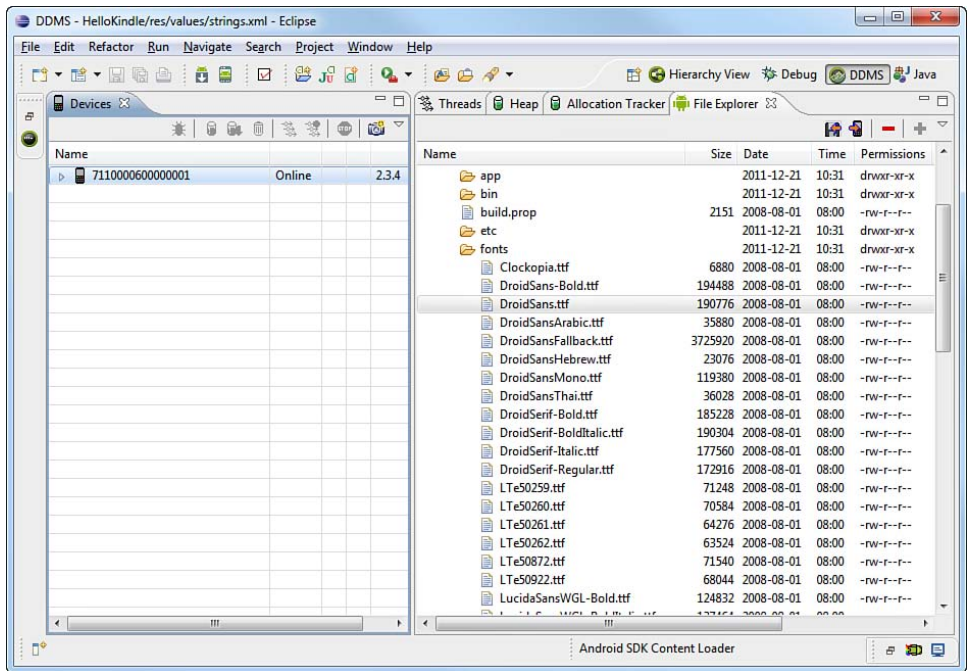



Figure 2.4 The File Explorer Tab in DDMS

Taking Screenshots of the Emulator or Device

One feature that can be particularly useful for debugging both devices and emulators is the ability to take screenshots of the current screen (see Figure 2.5).

The screenshot feature of the DDMS perspective is particularly useful when used with real devices. To take a screen capture of what's going on at this very moment on your device, follow these steps:

1. In the DDMS perspective, choose the device (or emulator) of which you want a screenshot. The device must be connected via USB.
2. On that device or emulator, make sure that you have the screen you want. Navigate to it, if necessary.
3. Click the Screen Capture button  to take a screen capture. This launches a Capture Screen dialog.
4. The Rotate button will rotate the Device Screen Capture tool to display in portrait mode. This is useful for Kindle Fire screen captures.
5. Within the Capture Screen dialog, click the Save button to save the screenshot to your local hard drive. This tool does not show a live view, just a snapshot; click the Refresh

button to update the capture view if you make changes on the device. The Copy button will place the image on your system's clipboard for pasting into another application, such as an image editor. Click the Done button to exit the tool and return to the DDMS perspective.

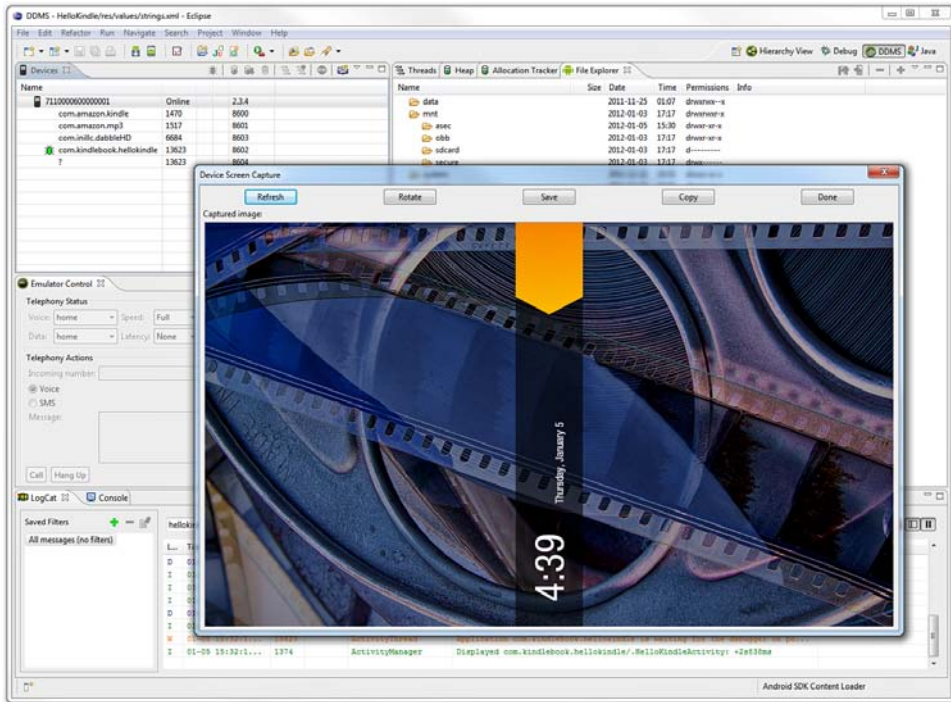


Figure 2.5 Taking a Screenshot Using DDMS

Viewing Log Information

The LogCat logging utility that is integrated into the DDMS perspective allows you to view the Android logging console. You may have noted the LogCat logging tab, with its diagnostic output, in Figure 2.2. We talk more about how to implement your own custom application logging in Chapter 3, “Building Kindle Fire Applications.”

Eclipse has the ability to filter logs by log severity. You can also create custom log filters by using tags. For more information on how to do this, see Appendix B, “Eclipse IDE Tips and Tricks.”

Working with the Android Emulator

The Android emulator is probably the most powerful tool at a developer's disposal. It is important for developers to learn to use the emulator and understand its limitations. The Android emulator is integrated with Eclipse, using the ADT plug-in for the Eclipse IDE.

The Android emulator is a convenient tool, but it has a number of limitations:

- The emulator is not a device. It simulates generic device behavior, not specific hardware implementations or limitations. This is particularly noticeable with Kindle Fire emulation; not even the home screen is the same. None of the custom Amazon Kindle Fire experience is emulated at this time.
- Sensor data, battery and power settings, and network connectivity are all simulated using your computer.
- No Kindle Fire built-in apps are present on the emulator.

Using the Android emulator is not a substitute for testing on a true Android device.

Providing Input to the Emulator

As a developer, you can provide input to the emulator in many ways:

- Use your computer mouse to click, scroll, and drag items (for example, sliding volume controls) onscreen and on the emulator skin.
- Use your computer keyboard to input text into controls.
- Use your mouse to simulate individual finger presses on the soft keyboard or physical emulator keyboard.
- Use a number of emulator keyboard commands to control specific emulator states.

Using Other Android Tools

Although we already covered the most important tools, a number of other special-purpose utilities are included with the Android SDK. A list of the tools that come as part of the Android SDK is available on the Android Developer website at <http://goo.gl/yzFHz>. Here, you can find a description of each tool and a link to its official documentation.

Summary

The Android SDK ships with a number of powerful tools to help with common Android development tasks. The Android documentation is an essential reference for developers. The DDMS debugging tool, which is integrated into the Eclipse development environment as a perspective, is useful for monitoring emulators and devices. The Android emulator can be used for running

and debugging Android applications, without the need for an actual device. There are also many other tools for interacting with physical devices and emulators in a variety of situations.

Exercises

1. Go to the Android Developer's Guide website at <http://d.android.com/guide>. Consider reading the article "Android Basics: What Is Android?" (<http://d.android.com/guide/basics/what-is-android.html>).
2. Launch the Android emulator. Get familiar with how the emulator tries to mimic a real Kindle Fire device. Note the limitations.
3. Launch the `HelloKindle` application you wrote in Chapter 1, "Getting Started with Kindle Fire," and explore it using the DDMS tool.

This page intentionally left blank

Building Kindle Fire Applications

Amazon built the Kindle Fire using the Android platform. Every platform technology uses different terminology to describe its application components. The three most important classes on the Android platform are `Context`, `Activity`, and `Intent`. Although there are other more advanced components that developers can implement, these three components form the building blocks for each and every Android application. In this chapter, we focus on understanding how Android applications are put together. We also look at some handy utility classes that can help developers debug applications.

Note

Some past readers assumed that they were to perform all the tasks discussed in this chapter on their own and build an app in one chapter without any help whatsoever. Not so! This chapter just gives you the 10,000-foot view of Android application development so that you have an idea of what you'll be implementing in future chapters. The one given here is simply a sample, not the one we'll build later, which specifically targets the Kindle Fire. We do this so you get an idea of how another application might be built, too. So, get yourself a cup of coffee, tea, or your "brain fuel" of choice, sit back, relax, and let's discuss the building blocks of Android apps.

Designing an Android Application

An Android application is a collection of tasks, each of which is called an activity. Each activity within an application has a unique purpose and user interface. To understand this more fully, imagine a theoretical game application called Chippy's Revenge.

Designing Application Features

The design of the Chippy's Revenge game is simple. It has five screens:

- **Splash**—This screen acts as a startup screen, with the game logo and version. It might also play some music.
- **Menu**—On this screen, a user can choose one of several options, including playing the game, viewing the scores, and reading the help text.
- **Play**—This screen is where game play actually takes place.
- **Scores**—This screen displays the highest scores for the game (including high scores from other players), providing players with a challenge to do better.
- **Help**—This screen displays instructions for how to play the game, including controls, goals, scoring methods, tips, and tricks.

Starting to sound familiar? You may recognize this generic design from many a mobile application, game or otherwise, on any platform.

Determining Application Activity Requirements

You need to implement five activity classes, one for each feature of the game:

- **SplashActivity**—This activity serves as the default activity to launch. It simply displays a layout (maybe just a big graphic), plays music for several seconds, and then launches **MenuActivity**.
- **MenuActivity**—This activity is straightforward. Its layout has several buttons, each corresponding to a feature of the application. The `onClick()` handlers for each button trigger cause the associated activity to launch.
- **PlayActivity**—The real application guts are implemented here. This activity needs to draw stuff onscreen, handle various types of user input, keep score, and generally follow whatever game dynamics the developer wants to support.
- **ScoresActivity**—This activity is about as simple as **SplashActivity**. It does little more than load a bunch of scoring information into a **TextView** control within its layout.
- **HelpActivity**—This activity is almost identical to **ScoresActivity**, except that instead of displaying scores, it displays help text. Its **TextView** control might possibly scroll.

Each activity class should have its own corresponding layout file stored in the application resources. You could use a single layout file for **ScoresActivity** and **HelpActivity**, but it's not necessary. If you did, however, you would simply create a single layout for both and set the image in the background and the text in the **TextView** control at runtime, instead of within the layout file.

Figure 3.1 shows the resulting design for your game, Chippy's Revenge Version 0.0.1 for Android.

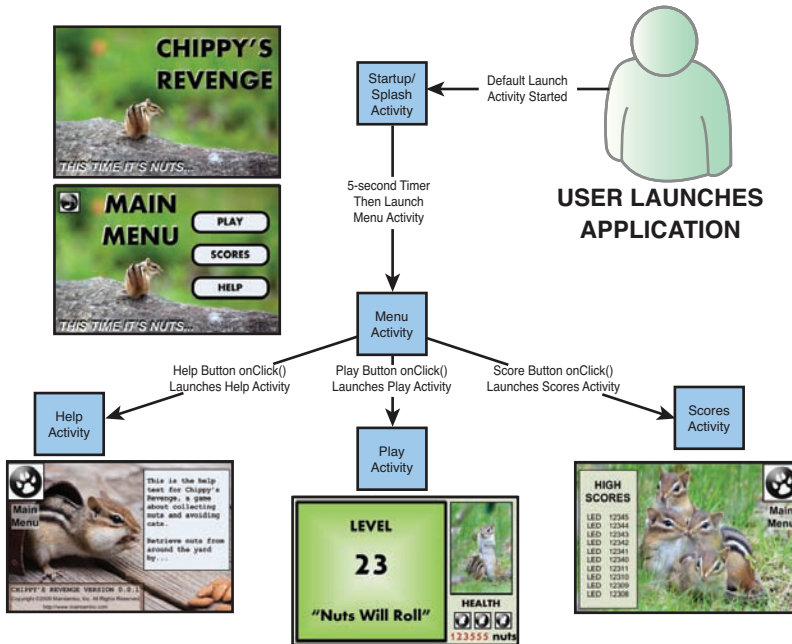


Figure 3.1 Chippy's Revenge Application Design

Implementing Application Functionality

Now that you understand how a typical Android application might be designed, you're probably wondering how to go about implementing that design.

We talked about how each activity has its own user interface, defined within a separate layout resource file. You might be wondering about implementation hurdles, such as the following:

- How do I control application state?
- How do I save settings?
- How do I display the user interface, defined within the layout resource file?
- How do I launch a specific activity?

With our theoretical game application in mind, it is time to dive into the implementation details of developing an Android application. A good place to start is the application context.

Using the Application Context

The application context is the central location for all top-level application functionality. You use the application context to access settings and resources shared across multiple activity instances.

You can retrieve the application context for the current process by using the `getApplicationContext()` method, like this:

```
Context context = getApplicationContext();
```

Because the `Activity` class is derived from the `Context` class, you can use the `this` object instead of retrieving the application context explicitly when you're writing code inside your `Activity` class.

After you retrieve a valid application context, you can use it to access application-wide features and services.

Retrieving Application Resources

You can retrieve application resources by using the `getResources()` method of the application context. The most straightforward way to retrieve a resource is by using its unique resource identifier, as defined in the automatically generated `R.java` class. The following example retrieves a `String` instance from the application resources by its resource ID:

```
String greeting = getResources().getString(R.string.hello);
```

Accessing Application Preferences

You can retrieve shared application preferences by using the `getSharedPreferences()` method of the application context. You can use the `SharedPreferences` class to save simple application data, such as configuration settings. Each `SharedPreferences` object can be given a name, which allows you to organize preferences into categories or store preferences all together in one large set.

For example, you might want to keep track of each user's name and some simple game state information, such as whether the user has credits left to play. The following code creates a set of shared preferences called `GamePrefs` and saves a few such preferences:

```
SharedPreferences settings = getSharedPreferences("GamePrefs", MODE_PRIVATE);
SharedPreferences.Editor prefEditor = settings.edit();
prefEditor.putString("UserName", "Spunky");
prefEditor.putBoolean("HasCredits", true);
prefEditor.commit();
```

To retrieve preference settings, you simply retrieve `SharedPreferences` and read the values back out:

```
SharedPreferences settings = getSharedPreferences("GamePrefs", MODE_PRIVATE);
String userName = settings.getString("UserName", "Chippy Jr. (Default)");
```

Accessing Other Application Functionality Using Contexts

The application context provides access to a number of top-level application features. Here are a few more things you can do with the application context:

- Launch `Activity` instances
- Retrieve assets packaged with the application
- Request a system-level service provider
- Manage private application files, directories, and databases
- Inspect and enforce application permissions

The first item on this list—launching `Activity` instances—is perhaps the most common reason you will use the application context.

Working with Activities

The `Activity` class is central to every Android application. Most of the time, you'll define and implement an activity for each screen in your application.

In the Chippy's Revenge game application, you have to implement five different `Activity` classes. In the course of playing the game, the user transitions from one activity to the next, interacting with the layout controls of each activity.

Launching Activities

There are a number of ways to launch an activity, including the following:

- Designating a launch activity in the manifest file
- Launching an activity using the application context
- Launching a child activity from a parent activity for a result

Designating a Launch Activity in the Manifest File

Each Android application must designate a default activity within the Android manifest file. If you inspect the manifest file of the `HelloKindle` project, you will notice that `HelloKindleActivity` is designated as the default activity.

In Chippy's Revenge, `SplashActivity` would be the most logical activity to launch by default.

Launching Activities Using the Application Context

The most common way to launch an activity is to use the `startActivity()` method of the application context. This method takes one parameter: an `Intent` object. We talk more about

the `Intent` class in the section “Working with Intents,” but for now, let’s look at a simple `startActivity()` call.

The following code calls the `startActivity()` method with an explicit intent:

```
startActivity(new Intent(getApplicationContext(), MenuActivity.class));
```

This intent requests the launch of the target activity, named `MenuActivity`, by its class. This class must be implemented elsewhere within the package. In fact, you could use this method to launch every activity in your theoretical game application; however, this is just one way to launch an activity.

The `MenuActivity` class must be registered within the Android manifest file before it can be launched as an activity.

Launching an Activity for a Result

Sometimes, you want to launch an activity, have it determine something (such as a user’s choice), and then return that information to the calling activity. When an activity needs a result, it can be launched using the `Activity.startActivityForResult()` method. The result will be returned in the `Intent` parameter of the calling activity’s `onActivityResult()` method. We talk more about how to pass data using an `Intent` parameter in a moment.

Managing Activity State

Applications can be interrupted when various higher priority events, such as alarms or certain types of notifications, take precedence. There can be only one active application at a time; specifically, a single application activity can be in the foreground at any given time. Although this is less common on a tablet, such as the Kindle Fire, you still need to be prepared for interruptions at any time based on user behavior, such as pressing the Home button and pausing the application.

Android applications are responsible for managing their state, as well as their memory, resources, and data. The Android operating system may terminate an activity that has been paused, stopped, or destroyed when memory is low. This means that any activity that is not in the foreground is subject to shut down. In other words, an Android application must keep state and be ready to be interrupted and even shut down at any time.

Using Activity Callbacks

The `Activity` class has a number of callbacks that provide an opportunity for an activity to respond to events, such as suspending and resuming. Table 3.1 lists the most important callback methods.

Table 3.1 Key Callback Methods of Android Activities

Callback Method	Description	Recommendations
<code>onCreate()</code>	Called when an activity starts or restarts	Initializes static activity data Binds to data or resources required Sets layout with <code>setContentView()</code>
<code>onResume()</code>	Called when an activity becomes the foreground activity	Acquires exclusive resources Starts any audio, video, or animations
<code>onPause()</code>	Called when an activity leaves the foreground	Saves uncommitted data Deactivates or releases exclusive resources Stops any audio, video, or animations
<code>onDestroy()</code>	Called when an application is shutting down	Cleans up any static activity data Releases any resources acquired.

The main thread is often called the UI thread, because this is where the processing for drawing the UI takes place internally. An activity must perform any processing that takes place during a callback reasonably quickly, so that the main thread is not blocked. If the main UI thread is blocked for too long, the Android system shuts down the activity because of a lack of response. It is especially important to respond quickly during the `onPause()` callback, when a higher priority task is entering the foreground.

Figure 3.2 shows the order in which activity callbacks are called.

Saving Activity State

An activity can have private preferences—much like shared application preferences. You can access these preferences by using the `getPreferences()` method of the activity. This mechanism is useful for saving state information. For example, `PlayActivity` for your game might use these preferences to keep track of the current level and score, player health statistics, and game state.

Shutting Down Activities

To shut down an activity, you make a call to the `finish()` method. There are several different versions of this method, depending on whether the activity is shutting itself down or shutting down another activity.

Within your game application, you might return from the Scores, Play, and Help screens to the Menu screen by finishing `ScoresActivity`, `PlayActivity`, or `HelpActivity`.

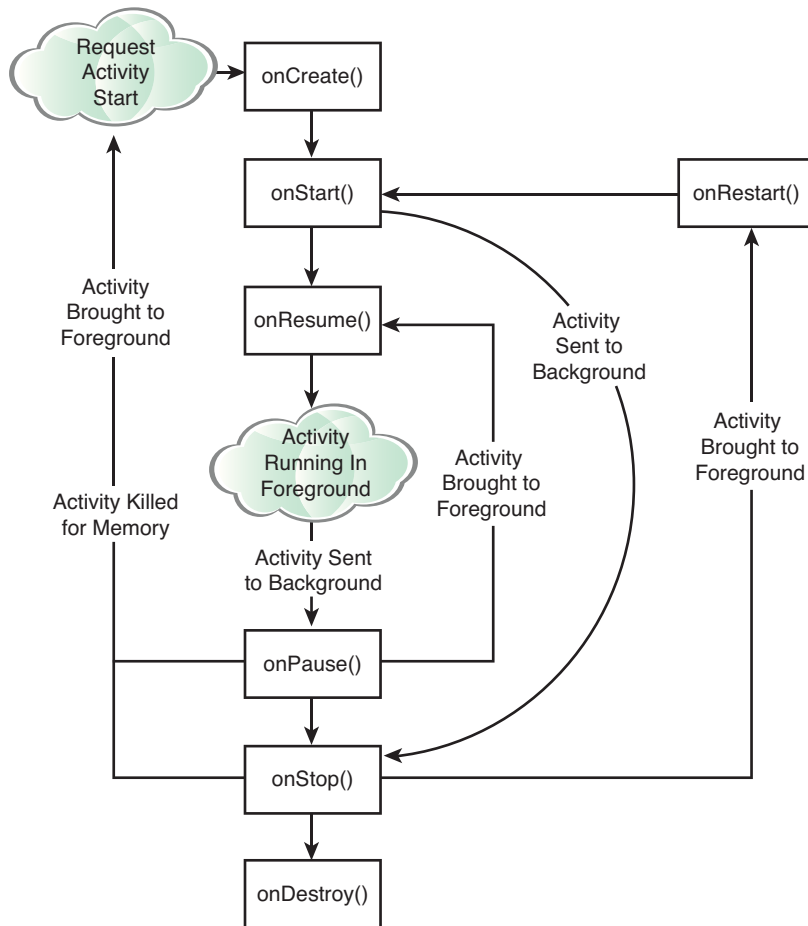


Figure 3.2 Important Activity Lifecycle Callbacks

Working with Intents

An `Intent` object encapsulates a task request used by the Android operating system. When the `startActivity()` method is called with the `Intent` parameter, the Android system matches the `Intent` action with appropriate activity on the Android system. That activity is then launched.

The Android system handles all intent resolution. An `Intent` instance can be very specific, including a request for a specific activity to be launched, or somewhat vague, requesting that any activity matching certain criteria be launched. For the details on intent resolution, see the Android documentation.

Passing Information with Intents

Intents can be used to pass data between activities. You can use an `Intent` object in this way by including additional data, called extras, within the intent.

To package extra pieces of data along with an intent, you use the `putExtra()` method with the appropriate type of object you want to include. The Android programming convention for intent extras is to name each one with the package prefix (for example, `com.androidbook.chippy.NameOfExtra`).

For example, the following intent includes an extra piece of information, the current game level, which is an integer:

```
Intent intent = new Intent(getApplicationContext(), HelpActivity.class);
intent.putExtra("com.androidbook.chippy.LEVEL", 23);
startActivity(intent);
```

When the `HelpActivity` class launches, the `getIntent()` method can be used to retrieve the intent. Then, the extra information can be extracted using the appropriate methods. Here's an example:

```
Intent callingIntent = getIntent();
int helpLevel = callingIntent.getIntExtra("com.androidbook.chippy.LEVEL", 1);
```

This little piece of information can be used to give special help hints, based on the level.

For the parent activity that launched a subactivity using the `startActivityForResult()` method, the result is passed in as a parameter to the `onActivityResult()` method with an `Intent` instance. The intent data can then be extracted and used by the parent activity.

Using Intents to Launch Other Applications

Initially, an application may only be launching activity classes defined within its own package. However, with the appropriate permissions, applications may also launch external activity classes in other applications.

There are well-defined intent actions for many common user tasks. For example, you can create intent actions to initiate applications such as the following:

- Launching the built-in web browser and supplying a URL address
- Launching the web browser and supplying a search string
- Launching the built-in email app and supplying a recipient, subject, and message body
- Launch third-party apps

Here is an example of how to create a simple intent with a predefined action (`ACTION_VIEW`) to launch the web browser with a specific URL:

```
Uri address = Uri.parse("http://www.perlgurl.org");
Intent surf = new Intent(Intent.ACTION_VIEW, address);
startActivity(surf);
```

This example shows an intent that has been created with an action and some data. The action, in this case, is to view something. The data is a uniform resource identifier (URI), which identifies the location of the resource to view.

For this example, the browser's activity then starts and comes into the foreground, causing the original calling activity to pause in the background. When the user finishes with the browser and taps the Back button, the original activity resumes.

Applications may also create their own intent types and allow other applications to call them, which makes for tightly integrated application suites.

Working with Dialogs

The Kindle Fire screen has more display space available than many types of applications might need. Instead of creating a whole new `Activity` to display a small amount of data, you may want to create a dialog instead. Dialogs can be helpful for creating simple user interfaces that do not necessitate an entirely new screen or activity to function. Instead, the calling activity dispatches a dialog, which can have its own layout and user interface, with buttons and input controls.

Table 3.2 lists the important methods for creating and managing activity dialog windows.

Table 3.2 Important Dialog Methods of the `Activity` Class

Method	Purpose
<code>Activity.showDialog()</code>	Shows a dialog, creating it if necessary.
<code>Activity.onCreateDialog()</code>	Is a callback when a dialog is being created for the first time and added to the activity dialog pool.
<code>Activity.onPrepareDialog()</code>	Is a callback for updating a dialog on-the-fly. Dialogs are created once and can be used many times by an activity. This callback enables the dialog to be updated just before it is shown for each <code>showDialog()</code> call.
<code>Activity.dismissDialog()</code>	Dismisses a dialog and returns to the activity. The dialog is still available to be used again by calling <code>showDialog()</code> again.
<code>Activity.removeDialog()</code>	Removes the dialog completely from the activity dialog pool.

Activity classes can include more than one dialog, and each dialog can be created and then used multiple times.

There are many ready-made dialog types available for use in addition to the basic dialog. These are `AlertDialog`, `CharacterPickerDialog`, `DatePickerDialog`, `ProgressDialog`, and `TimePickerDialog`.

You can also create an entirely custom dialog by designing an XML layout file and using the `Dialog setContentView()` method. To retrieve controls from the dialog layout, you simply use the `Dialog findViewById()` method.

Dialogs and Dialog Fragments

You may have seen in the Android SDK documentation that these `Activity` dialog-related methods are deprecated in favor of something called a fragment. Fragments were introduced in API Level 11 (Android 3.0+). At the time of this writing, the Kindle Fire runs API Level 10 and, as such, does not have `Fragment` classes available. That being said, the Android Support package (a separate JAR file that can be linked to your project and is included with the SDK) can be used to gain access to some of these SDK classes that were introduced in later API levels, including `DialogFragment`, in your Kindle Fire applications. You might do this if you're porting a newer application to the Kindle Fire platform or writing an application that you plan on using on other tablets.

Working with Fragments

The concept of fragments is relatively new to Android. A fragment is simply a block of UI, with its own lifecycle, that can be reused within different activities. Fragments allow developers to create highly modular user-interface components that can change dramatically based on screen sizes, orientation, and other aspects of the display that might be relevant to the design.

Table 3.3 shows some important lifecycle calls that are sent to the `Fragment` class.

Table 3.3 Key Fragment Lifecycle Callbacks

Method	Purpose
<code>onCreateView()</code>	Called when the fragment needs to create its view
<code>onStart()</code>	Called when the fragment is made visible to the user
<code>onPause()</code>	Similar to <code>Activity.onPause()</code>
<code>onStop()</code>	Called when the fragment is no longer visible
<code>onDestroy()</code>	Final fragment cleanup

Although the lifecycle of a fragment is similar to that of an activity, a fragment only exists within an activity. A common example of fragment usage is to change the UI flow between portrait and landscape modes. If an interface has a list of items and a details view, the list and the details could both be fragments. In portrait orientation, the screen would show the list view followed by the details view, both full screen. But, in landscape mode, the view could show the list and details side by side.

The modular nature of fragments makes them a powerful user-interface building block. For use on the Kindle Fire platform, much of the `Fragment` API is available via the Android Support library.

Logging Application Information

Android provides a useful logging utility class called `android.util.Log`. Logging messages are categorized by severity (and verbosity), with errors being the most severe. Table 3.4 lists some commonly used logging methods of the `Log` class.

Table 3.4 Commonly Used Log Methods

Method	Purpose
<code>Log.e()</code>	Logs errors
<code>Log.w()</code>	Logs warnings
<code>Log.i()</code>	Logs informational messages
<code>Log.d()</code>	Logs debug messages
<code>Log.v()</code>	Logs verbose messages
<code>Log.wtf()</code>	Logs messages for events that should not happen (like during a failed assert)

The first parameter of each `Log` method is a string called a tag. One common Android programming practice is to define a global static string to represent the overall application or the specific activity within the application such that log filters can be created to limit the log output to specific data.

For example, you could define a string called `TAG`, as follows:

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

Now, anytime you use a `Log` method, you supply this tag. An informational logging message might look like this:

```
Log.i(TAG, "In onCreate() callback method");
```

You can use the `LogCat` utility from within Eclipse to filter your log messages to the tag string. See Appendix B, “Eclipse IDE Tips and Tricks,” for details.

Note

Excessive use of the `Log` utility can result in decreased application performance. Debug and verbose logging should be used only for development purposes and removed before application publication. While naming of the tag used in logging is arbitrary, standard practice is to use the class name or activity name.

Summary

In this chapter, you saw how different Android applications can be designed using three application components: `Context`, `Activity`, and `Intent`. Each Android application comprises one or more activities. Top-level application functionality is accessible through the application context. Each activity has a special function and (usually) its own layout or user interface. An activity is launched when the Android system matches an intent object with the most appropriate application activity, based on the action and data information set in the intent. Intents can also be used to pass data from one activity to another.

In addition to learning the basics of how Android applications are put together, you also learned how to take advantage of useful Android utility classes, such as application logging, which can help streamline Android application development and debugging.

Exercises

1. Add a logging tag to the `HelloKindleActivity` class you created in the `HelloKindle` project in Chapter 1, “Getting Started with Kindle Fire.” Within the `onCreate()` callback method, add an informational logging message, using the `Log.i()` method. Run the application and view the log output in the Eclipse DDMS or Debug perspectives within the LogCat tab.
2. Within the `HelloKindleActivity` class you created in the `HelloKindle` project in Chapter 1, add method stubs for the `Activity` callback methods in addition to `onCreate()`, such as `onStart()`, `onRestart()`, `onResume()`, `onPause()`, `onStop()`, and `onDestroy()`. To do this easily from within Eclipse, right-click the `HelloKindleActivity.java` class and choose `Source, Override/Implement methods`. Under the `Activity` class methods, select the methods just suggested and click the OK button. You will see appropriate method stubs added for each method you selected.
3. Add a log message to each `Activity` class callback method you created in exercise 2. For example, add an informational log message, such as `In` method `onCreate()` to the `onCreate()` method. Run the application normally and view the log output to trace the application lifecycle. Next, try some other scenarios, such as pausing or suspending the application and then resuming. Simulate an incoming call using the Eclipse DDMS perspective while running your application and see what happens.

This page intentionally left blank

Managing Application Resources

Android applications for the Kindle Fire rely on strings, graphics, and other types of resources to generate robust user interfaces. Android projects can include these resources, using a well-defined project resource hierarchy. In this chapter, we review the most common types of resources used by Android applications, how they are stored, and how they can be accessed programmatically. This chapter prepares you for working with resources in future chapters, but you are not directly asked to write code or create resources.

Using Application and System Resources

Resources are broken down into two types: application resources and system resources. Application resources are defined by the developer within the Android project files and are specific to the application. System resources are common resources defined by the Android platform and accessible to all applications through the Android SDK. You can access both types of resources at runtime.

You can load resources in your Java code, usually from within an activity. You can also reference resources from within other resources; for example, you might reference numerous string, dimension, and color resources from inside an XML layout resource to define the properties and attributes of specific controls, like background colors and text to display.

Working with Application Resources

Application resources are created and stored within the Android project files under the `/res` directory. Using a well-defined but flexible directory structure, resources are organized, defined, and compiled with the application package. Application resources are not shared with the rest of the Android system.

Storing Application Resources

Defining application data as resources (as opposed to at runtime in code) is good programming practice. Grouping application resources together and compiling them into the application package has the following benefits:

- Code is cleaner and easier to read, which leads to fewer bugs.
- Resources are organized by type and guaranteed to be unique.
- Localization and internationalization are straightforward.

The Android platform supports a variety of resource types (see Figure 4.1), which can be combined to form different types of applications. The Kindle Fire device is no different, although its screen is larger than most Android smartphones and smaller than many tablets.

Android applications can include many different kinds of resources. The following are some of the most common resource types:

- Strings, colors, and dimensions
- Drawable graphics files
- Layout files
- Raw files of all types

Resource types are defined with special XML tags and organized into specially named project directories. Some `/res` subdirectories, such as the `/drawable`, `/layout`, and `/values` directories, are created by default when a new Android project is created, while others must be added by the developer when required.

Resource files stored within `/res` subdirectories must abide by the following rules:

- Resource filenames must be lowercase.
- Resource filenames may only contain letters, numbers, underscores, and periods.
- Resource filenames (and XML name attributes) must be unique.

When resources are handled during the build process, their name dictates their variable name. For example, a graphics file saved within the `/drawable` directory as `mypic.jpg` is referenced as `@drawable/mypic`. It is important to name resource names intelligently and be aware of character limitations that are stricter than file-system names. (For example, dashes cannot be used in image filenames.) Consult the Android documentation for specific project directory naming conventions.

Each time you save a resource file (that is, copy a resource file such as a graphics file into the appropriate directory) within Eclipse, the `R.java` class file is recompiled to incorporate your changes. If you have not used the correct directory-naming or filenaming conventions, you see a compiler error in the Eclipse Problems tab. (This assumes that the Eclipse default setting for Build Automatically is set in the Project menu.)

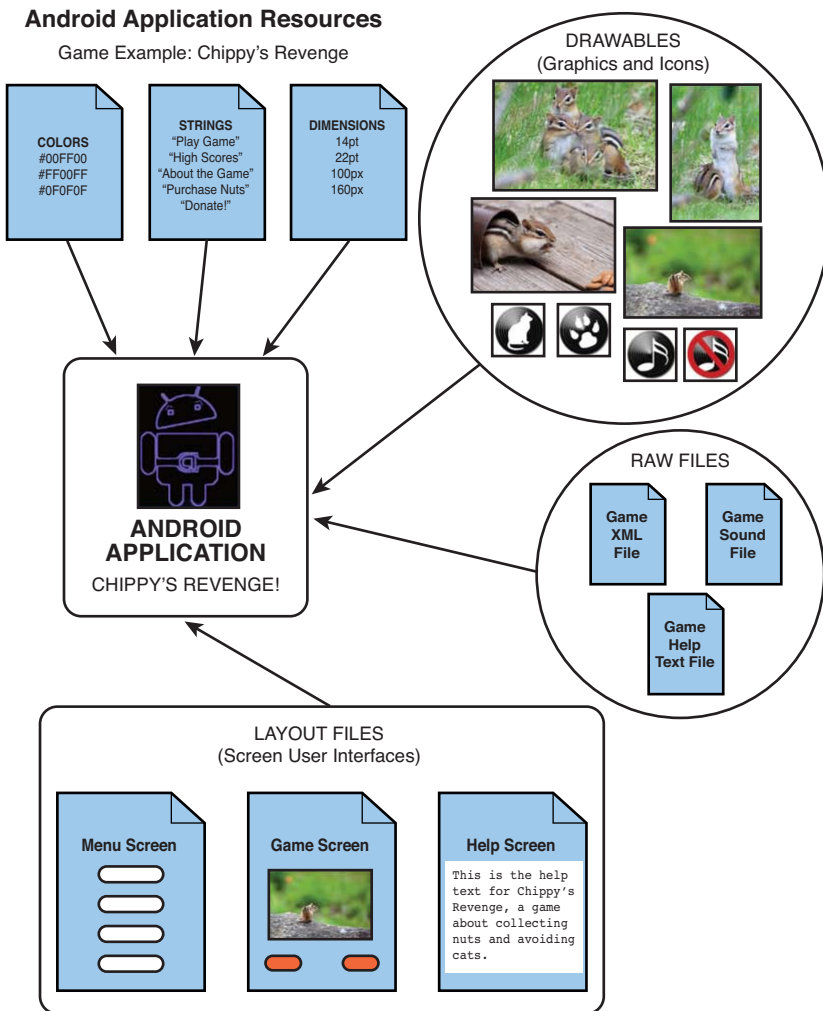


Figure 4.1 Android Applications Can Use a Variety of Resources

Referencing Application Resources

All application resources are stored within the `/res` project directory structure and are compiled into the project at build time. Application resources can be used programmatically. They can also be referenced in other application resources.

Application resources can be accessed programmatically using the generated class file called `R.java`. To reference a resource from within your `Activity` class, you must retrieve the

application's `Resources` object using the `getResources()` method and then make the appropriate method call based on the type of resource you want to retrieve.

For example, to retrieve a string named `hello` defined in the `strings.xml` resource file, use the following method call:

```
String greeting = getResources().getString(R.string.hello);
```

We talk more about how to access different types of resources later in this chapter.

To reference an application resource from another compiled resource, such as a layout file, use the following format:

```
@[resource type]/[resource name]
```

For example, the same string used earlier would be referenced as follows:

```
@string/hello
```

We talk more about referencing resources later in this chapter, when we discuss layout files.

Working with System Resources

Applications can access the Android system resources in addition to their private resources. This “standardized” set of resources is shared across all applications, providing users with common styles and other useful templates, as well as commonly used strings and colors.

System resources are stored within the `android.R` package. There are classes for each major resource type. For example, the `android.R.string` class contains the system string resources. For example, to retrieve a system resource string called `ok` from within an `Activity` class, you first need to use the static method of the `Resources` class called `getSystem()` to retrieve the global system `Resource` object. Then, you call the `getString()` method with the appropriate string resource name, like this:

```
String confirm = Resources.getSystem().getString(android.R.string.ok);
```

To reference a system resource from another compiled resource, such as a layout resource file, use the following format:

```
@android:[resource type]/[resource name]
```

For example, you could use the system string for `ok` by setting the appropriate string attribute as follows:

```
@android:string/ok
```

System Resources on the Kindle Fire Device

System resources are available on the Kindle Fire, just like any Android device. The resources provided are a fairly typical set. Curious about which system resources are available on your Kindle Fire and what they look like? Check out the app called `rs:ResEnum`. Written by a developer for developers, this app enumerates all system resources and displays the drawable ones.

Working with Simple Resource Values

Simple resources, such as string, color, and dimension values, should be defined in XML files under the `/res/values` project directory. These resource files use special XML tags that represent name/value pairs. These types of resources are compiled into the application package at build time. You can manage string, color, and dimension resources by using the Eclipse resource editor, or you can directly edit the XML resource files.

Working with Strings

You can use string resources anywhere your application needs to display text. You define string resources with the `<string>` tag, identify them with the `name` property, and store them in the resource file `/res/values/strings.xml`.

Here is an example of a string resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Name this App</string>
    <string name="hello">Hello</string>
</resources>
```

String resources have a number of formatting options. Strings that contain apostrophes or single straight quotes must be escaped or wrapped within double straight quotes. Table 4.1 shows some simple examples of well-formatted string values.

Table 4.1 String Resource Formatting Examples

String Resource Value	Will Be Displayed As
Hello, World	Hello, World
"Hello, World"	Hello, World
Mother\'s Maiden Name:	Mother's Maiden Name:
He said, \"No.\"	He said, "No."

There are several ways to access a string resource programmatically. The simplest way is to use the `getString()` method within your `Activity` class:

```
String greeting = getResources().getString(R.string.hello);
```

Working with Colors

You can apply color resources to screen controls. You define color resources with the `<color>` tag, identify them with the `name` attribute, and store them in the file `/res/values/colors.xml`. This XML resource file is not created by default; it must be created manually.

You can add a new XML file, such as this one, by choosing File, New, Android XML File, and then fill out the resulting dialog with the type of file (such as values). This automatically sets the expected folder and type of file for the Android project.

Here is an example of a color resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="background_color">#006400</color>
    <color name="app_text_color">#FFE4C4</color>
</resources>
```

The Android system supports 12-bit and 24-bit colors in RGB format. Table 4.2 lists the color formats that the Android platform supports.

Table 4.2 Color Formats Supported in Android

Format	Description	Example
#RGB	12-bit color	#00F (blue)
#ARGB	12-bit color with alpha	#800F (blue, alpha 50%)
#RRGGBB	24-bit color	#FF00FF (magenta)
#AARRGGBB	24-bit color with alpha	#80FF00FF (magenta, alpha 50%)

The following Activity class code snippet retrieves a color resource named `app_text_color` using the `getColor()` method:

```
int textColor = getResources().getColor(R.color.app_text_color);
```

Working with Dimensions

To specify the size of a user interface control, such as a `Button` or `TextView` control, you need to specify different kinds of dimensions. Dimension resources are helpful for font sizes, image sizes, and other physical or pixel-relative measurements. You define dimension resources with the `<dimen>` tag, identify them with the `name` property, and store them in the resource file `/res/values/dimens.xml`. This XML resource file is not created by default and must be created manually.

Here is an example of a dimension resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="thumbDim">100px</dimen>
</resources>
```

Each dimension resource value must end with a unit of measurement. Table 4.3 lists the dimension units that Android supports.

Table 4.3 Dimension Unit Measurements Supported in Android

Type of Measurement	Description	Unit String
Pixels	Actual screen pixels	px
Inches	Physical measurement	in
Millimeters	Physical measurement	mm
Points	Common font measurement	pt
Density-independent pixels	Pixels relative to 160dpi	dp
Scale-independent pixels	Best for scalable font display	sp

The following `Activity` class code snippet retrieves a dimension resource called `thumbDim` using the `getDimension()` method:

```
float thumbnailDim = getResources().getDimension(R.dimen.thumbDim);
```

Working with Drawable Resources

Drawable resources, such as image files, must be saved under the `/res/drawable` project directory hierarchy. Typically, applications provide multiple versions of the same graphics for different pixel density screens. A default Android project contains three drawable directories: `drawable-ldpi` (low density), `drawable-mdpi` (medium density), and `drawable-hdpi` (high density). There are many more possible drawable directories. The system picks the correct version of the resource based on the device on which the application is running. Kindle Fire drawable resources should be stored in the `mdpi` directory. All versions of a specific resource must have the same name in each of the drawable directories. These types of resources are then compiled into the application package at build time and are available to the application.

You can drag and drop image files into the `/res/drawable` directories by using the Eclipse Project Explorer. Again, remember that filenames must be unique within a particular drawable directory, lowercase, and contain only letters, numbers, and underscores.

Working with Images

The most common drawable resources used in applications are bitmap-style image files, such as PNG and JPG files. These files are often used as application icons and button graphics but may be used for a number of user interface components.

As shown in Table 4.4, Android supports many common image formats.

Table 4.4 Image Formats Supported in Android

Supported Image Format	Description	Required Extension
Portable Network Graphics	Preferred format (lossless)	.png (PNG)
Nine-Patch Stretchable Images	Preferred format (lossless)	.9.png (PNG)
Joint Photographic Experts Group	Acceptable format (lossy)	.jpg (JPEG/JPG)
Graphics Interchange Format	Discouraged but supported (lossless)	.gif (GIF)

Using Image Resources Programmatically

Image resources are encapsulated in the class `BitmapDrawable`. To access a graphic resource file called `/res/drawable/logo.png` within an `Activity` class, you would use the `getDrawable()` method, as follows:

```
BitmapDrawable logoBitmap =
    (BitmapDrawable) getResources().getDrawable(R.drawable.logo);
```

Most of the time, however, you don't need to load a graphic directly. Instead, you can use the resource identifier as the source attribute on a control such as an `ImageView` control within a compiled layout resource and it will be displayed on the screen. However, there are times when you might want to programmatically load, process, and set the drawable for a given `ImageView` control at runtime. The following `Activity` class code sets and loads the `logo.png` drawable resource into an `ImageView` control named `LogoImageView`, which must be defined in advance:

```
ImageView logoView = (ImageView) findViewById(R.id.LogoImageView);
logoView.setImageResource(R.drawable.logo);
```

Working with Other Types of Drawables

In addition to graphics files, you can create specially formatted XML files to describe other `Drawable` subclasses, such as `ShapeDrawable`. You can use the `ShapeDrawable` class to define different shapes, such as rectangles and ovals. See the Android documentation for the `android.graphics.drawable` package for further information.

Working with Layouts

Most Android application user interface screens are defined using specially formatted XML files called layouts. Layout XML files can be considered a special type of resource: They are generally used to define what a portion of, or all of, the screen will look like. It can be helpful to think of

a layout resource as a template; you fill a layout resource with different types of view controls, which may reference other resources, such as strings, colors, dimensions, and drawables.

In truth, layouts can be compiled into the application package as XML resources or be created at runtime in Java from within your `Activity` class using the appropriate layout classes within the Android SDK. However, in most cases, using the XML layout resource files greatly improves the clarity, readability, and reusability of code and flexibility of your application.

Layout resource files are stored in the `/res/layout` directory hierarchy. You compile layout resources into your application as you would any other resources.

Here is an example of a layout resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
</LinearLayout>
```

You might recognize this layout: It is the default layout, called `main.xml`, created with any new Android application. This layout file describes the user interface of the only activity within the application. It contains a `LinearLayout` control that is used as a container for all other user interface controls—in this case, a single `TextView` control. The `main.xml` layout file also references another resource: the string resource called `@string/hello`, which is defined in the `strings.xml` resource file.

As with the multiple drawable directories, there can be multiple layout directories: in particular, `/layout-land` for landscape files and `/layout-port` for portrait files. As with image drawables, the files must be named the same. This gives you the ability to have a different layout file for landscape and portrait orientations that the Kindle Fire supports.

Designing Layouts Using the Layout Resource Editor

You can design and preview compiled layout resources in Eclipse by using the layout resource editor (see Figure 4.2). Double-click the project file `/res/layout/main.xml` within Eclipse to launch the layout resource editor. The layout resource editor has two tabs: Graphical Layout and `main.xml`. The Graphical Layout tab provides drag-and-drop visual design and the ability to preview the layout in various device configurations. You can configure one for Kindle Fire using the custom option, if you like. The `main.xml` tab allows you to directly edit the layout XML.

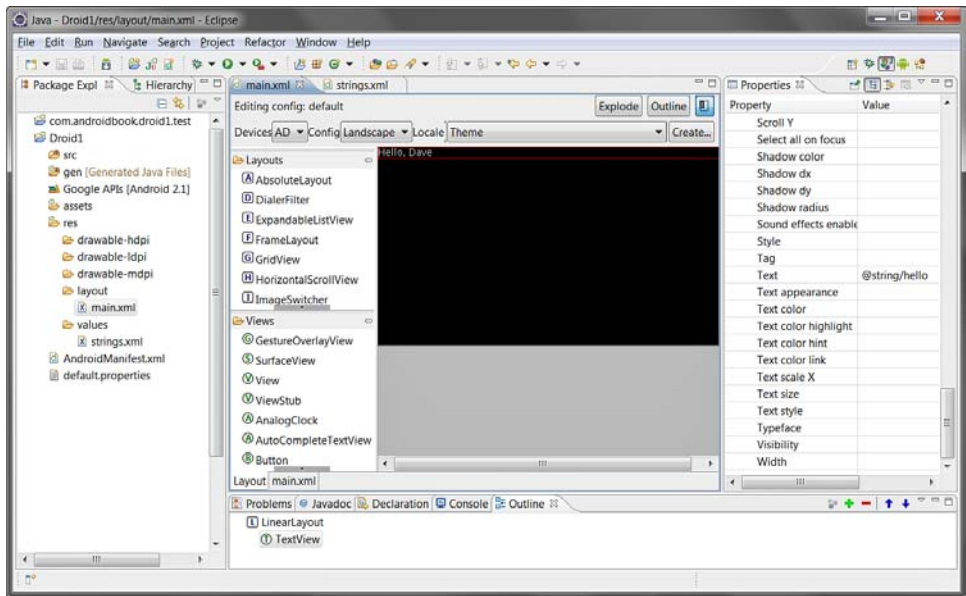


Figure 4.2 The Layout Resource Editor in Eclipse

Chances are, you'll switch back and forth between the Graphical Layout tool and XML tabs frequently. There are also several other Eclipse panes that are helpful for using with the layout resource editor: the Outline pane and the Properties pane. You can add and remove controls to the specific layout using the Outline pane (see Figure 4.2, bottom). You can set individual properties and attributes of a specific control by using the Properties pane (see Figure 4.2, right). Note that Eclipse panes are not fixed—drag them around and configure them in a way that works for you. Eclipse actually calls these panes “views” (which might be confusing for Android folks). You can also add different types of view “panes” from the Windows menu of Eclipse.

Like most other user-interface designers, the layout resource editor works well for basic layout design, but it has some limitations. For some of the more complex user interface controls, you might be forced to edit the XML by hand. You might also lose the ability to preview your layout if you add a control to your layout that is not supported by the Graphical Layout tool. In such a case, you can still view your layout by running your application in the emulator or on your Kindle Fire. Displaying an application correctly on a Kindle Fire, rather than the Eclipse layout editor, should always be your primary objective.

Designing Layouts Using XML

You can edit the raw XML of a layout file. As you gain experience developing layouts, you should familiarize yourself with the XML layout file format. Switch to the XML view frequently and accustom yourself to the XML generated by each type of control. Do not rely on the

Graphical Layout tool alone—equivalent to a web designer who knows how to use a web-design tool but doesn't know HTML. Although the Graphical Layout tool has gotten much better in recent times, knowing the XML can help debug tricky problems.

Tired of just theory? Give the Eclipse layout editor a spin:

1. Open the `HelloKindle` Android project you created in Chapter 1, “Getting Started with Kindle Fire.”
2. Navigate to the `/res/layout/main.xml` layout file and double-click the file to open it in the Eclipse layout resource editor.
3. Switch to the Graphical Layout tab, and you should see the layout preview in the main window.
4. Click the Outline tab. This pane displays the view control hierarchy of XML elements in this layout resource. In this case, you have a `LinearLayout` control. If you expand it, you see that it contains a `TextView` control.
5. Select the `TextView` control on the Outline tab. You see a colored box highlight the `TextView` control in the layout preview.
6. Click the Properties tab. This tab displays all the properties and attributes that can be configured for the `TextView` control you just selected. Scroll down to the property called `Text`, and note that it has been set to a string resource called `@string/hello`.
7. Click the `Text` property called `@string/hello` on the Properties tab. You can now modify the field. You can directly type in a string, manually enter a different string resource (`@string/app_name`, for example), or click the little button with the three dots and choose an appropriate resource from the list of string resources available to your application. Each time you change this field, note how the Graphical Layout preview is updated automatically.
8. Switch to the `main.xml` tab and note how the XML is structured. Changes you make in the XML tab are immediately reflected in the Graphical Layout tab. If you save and run your project in the emulator, you should see results similar to those displayed in the preview.

Feel free to continue to explore the layout resource editor. You might want to try adding additional view controls, such as an `ImageView` control or another `TextView` control, to your layout. We cover designing layouts in more detail later in this book.

Using Layout Resources Programmatically

Layout controls, whether `Button`, `ImageView`, `TextView`, or `LinearLayout`, are derived from the `View` class. In most instances, you do not need to load and access a whole layout resource programmatically. Instead, you simply want to modify specific view controls within it. For example, you might want to change the text being displayed by the `TextView` control in the `main.xml` layout resource.

The default layout file created with the `HelloKindle` project contains one `TextView` control. However, this `TextView` control does not have a `name` attribute. The easiest way to access the correct `View` control is by its unique name, so take a moment and set the `id` attribute of the `TextView` control using the layout resource editor. Call it `@+id/TextView01`.

Now that your `TextView` control has a unique identifier, you can find it from within your `Activity` class using the `findViewById()` method. After you find the `TextView` you are looking for, you are free to call its methods, such as the `TextView` class's `setText()` method. Here's how you would retrieve a `TextView` object named `TextView01` that has been defined in the layout resource file:

```
TextView txt = (TextView)findViewById(R.id.TextView01);
```

Note that the `findViewById()` method takes a resource identifier—the same one you just configured in your layout resource file. Here's what's happening behind the scenes: When you save the layout resource file as XML, Eclipse automatically recompiles the generated `R.java` file associated with your project, making the identifier available for use within your Java classes. (If you don't have the `Build Automatically` setting in the `Project` menu turned on, you have to build the project manually.)

Working with Files

In addition to string, graphic, and layout resources, Android projects can contain files as resources. These files may be in any format. However, some formats are more convenient than others.

Working with XML Files

As you might expect, the XML file format is well supported on the Android platform. Arbitrary XML files can be included as resources. These XML files are stored in the `/res/xml` resource directory. XML file resources are the preferred format for any structured data your application requires.

How you format your XML resource files is up to you. A variety of XML utilities are available as part of the Android platform, as shown in Table 4.5.

Table 4.5 XML Utility Packages

Package	Description
<code>android.sax.*</code>	Framework to write standard SAX handlers
<code>android.util.Xml.*</code>	XML utilities, including the <code>XMLPullParser</code>
<code>org.xml.sax.*</code>	Core SAX functionality (see http://www.saxproject.org)
<code>javax.xml.*</code>	SAX and limited DOM, Level 2 core support

Package	Description
org.w3c.dom	Interfaces for DOM, Level 2 core
org.xmlpull.*	XmlPullParser and XMLSerializer interfaces (see http://www.xmlpull.org)

To access an XML resource file called `/res/xml/default_values.xml` programmatically from within your `Activity` class, you could use the `getXml()` method of the `Resources` class, like this:

```
XmlResourceParser defaultDataConfig = getResources().getXml(R.xml.default_values);
```

After you had access to an XML parser, you could parse your XML, extract the appropriate data, and do with it whatever you want.

Working with Raw Files

An application can include raw files as resources. Raw files your application might use include audio files, video files, and any other file formats you might need. All raw resource files should be included in the `/res/raw` resource directory. All raw file resources must have unique names, excluding the file suffix (meaning that `file1.txt` and `file1.dat` would conflict).

If you plan to include media file resources, consult the Android platform documentation and Amazon's Kindle Fire FAQ to determine what media formats and encodings are supported. A general list of supported formats for Android devices is available at <http://goo.gl/wMNS9>, while the list for the Kindle Fire is at <http://goo.gl/hNRnX>.

The same goes for any other file format you want to include as an application resource. If the file format you plan on using is not supported by the native Android system, your application will be required to do all file processing itself.

To access a raw file resource programmatically from within your `Activity` class, simply use the `openRawResource()` method of the `Resources` class. For example, the following code creates an `InputStream` object to access the resource file `/res/raw/file1.txt`:

```
InputStream iFile = getResources().openRawResource(R.raw.file1);
```

Note

There are times when you might want to include files within your application but not have them compiled into application resources. Android provides a special project directory called `/assets` for this purpose. This project directory resides at the same level as the `/res` directory. Any files included in this directory are included as binary resources, along with the application installation package, and are not compiled into the application.

Uncompiled files, called application assets, are not accessible through the `getResources()` method. Instead, you must use `AssetManager` to access files included in the `/assets` directory.

Working with Other Types of Resources

We covered the most common types of resources that you might need in an application. There are also numerous other types of resources available. These resource types may be used less often and be more complex. However, they allow for very powerful applications. Some of the other types of resources you can take advantage of include the following:

- Primitives (boolean values, integer values)
- Arrays (string arrays, integer arrays, typed arrays)
- Menus
- Animation sequences
- Shape drawables
- Styles and themes
- Custom layout controls

When you are ready to use these other resource types, consult the Android documentation for further details. <http://goo.gl/X9XZj> is a good place to start.

Summary

Kindle Fire applications can use many different types of resources, including application-specific resources and system-wide resources. The Eclipse resource editors facilitate resource management, but XML resource files can also be edited manually. Once defined, resources can be accessed programmatically, as well as referenced by name by other resources. String, color, and dimension values are stored in specially formatted XML files, and graphic images are stored as individual files. Application user interfaces are defined using XML layout files. Raw files, which can include custom data formats, may also be included as resources for use by the application. Finally, applications may include numerous other types of resources as part of their packages.

Exercises

1. Add a new color resource with a value of `#00ff00` to your `HelloKindle` project. Within the `main.xml` layout file, use the Properties pane to change the `textColor` attribute of the `TextView` control to the color resource you just created. View the layout in the Eclipse layout resource editor, rerun the application, and view the result on an emulator or Kindle Fire—in all three cases, you should see green text.
2. Add a new dimension resource with a value of `22pt` to your `HelloKindle` project. Within the `main.xml` layout file, use the Properties pane to change the `textSize` attribute of the `TextView` control to the dimension resource you just created. View the

layout in the Eclipse layout resource editor, rerun the application, and view the result on an emulator or Kindle Fire—in all three cases, you should see larger font text (22pt).

3. Add a new drawable graphics file resource to your `HelloKindle` project (e.g., a small .png or .jpg file). Within the `main.xml` layout resource file, use the Outline pane to add an `ImageView` control to the layout. Then, use the Properties pane to set the `ImageView` control's `src` attribute to the drawable resource you just created. View the layout in the Eclipse layout resource editor, rerun the application, and view the result on an emulator or Kindle Fire—in all three cases, you should see an image below the text on the screen.

This page intentionally left blank

Configuring the Android Manifest File

Every Android project, for the Kindle Fire or otherwise, includes a special file called the Android manifest file. The Android system uses this file to determine application configuration settings, including the application's identity as well as what permissions the application requires to run. In this chapter, we examine the Android manifest file and look at how different applications use its features.

Exploring the Android Manifest File

The Android manifest file, named `AndroidManifest.xml`, is an XML file that must be included at the top level of any Android project. If you use Eclipse with the ADT plug-in, the Android project wizard will create the initial `AndroidManifest.xml` file with default values for the most important configuration settings. The Android system uses the information in this file to do the following:

- Install and upgrade the application package
- Display application details to users
- Launch application activities
- Manage application permissions
- Handle a number of other advanced application configurations, including acting as a service provider or content provider

You can edit the Android manifest file by using the Eclipse manifest file resource editor or by manually editing the XML.

The Eclipse manifest file resource editor organizes the manifest information into categories presented on five tabs:

- Manifest
- Application
- Permissions
- Instrumentation
- AndroidManifest.xml

Using the Manifest Tab

The Manifest tab (see Figure 5.1) contains package-wide settings, including the package name, version information, and minimum Android SDK version information. You can also set any hardware configuration requirements here.

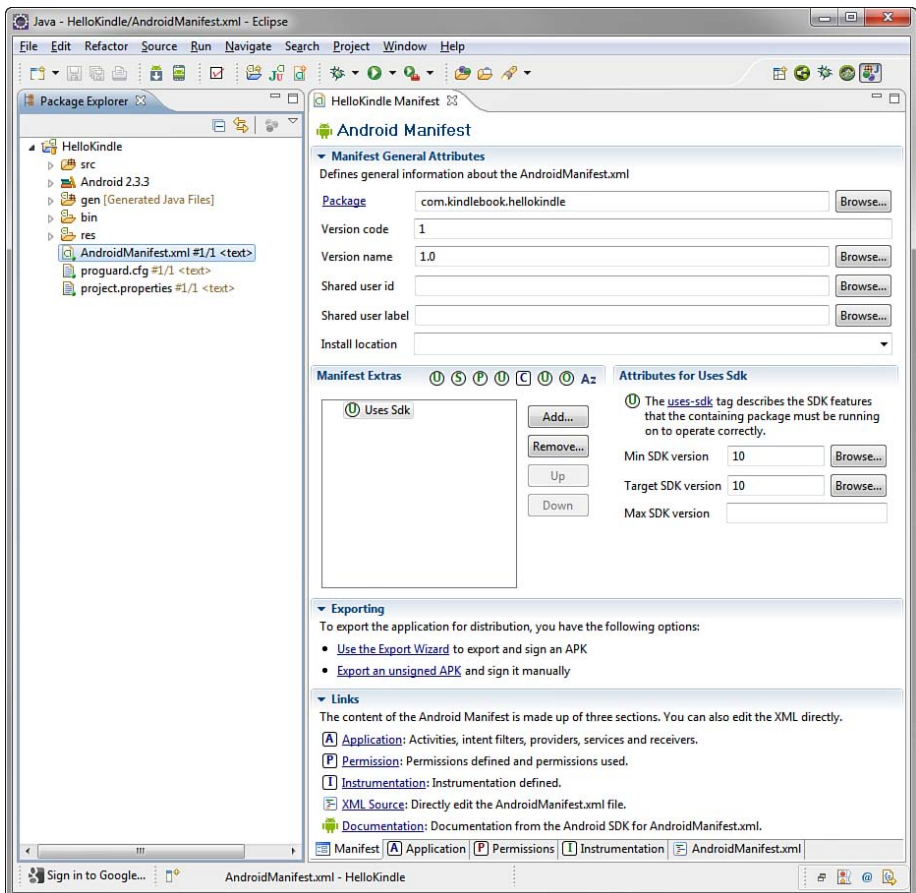


Figure 5.1 The Manifest Tab of the Eclipse Manifest File Resource Editor

Using the Application Tab

The Application tab (see Figure 5.2) contains application-wide settings, including the application label and icon, as well as information about application components, such as activities, intent filters, and other application functionality, including configuration for service and content provider implementations.

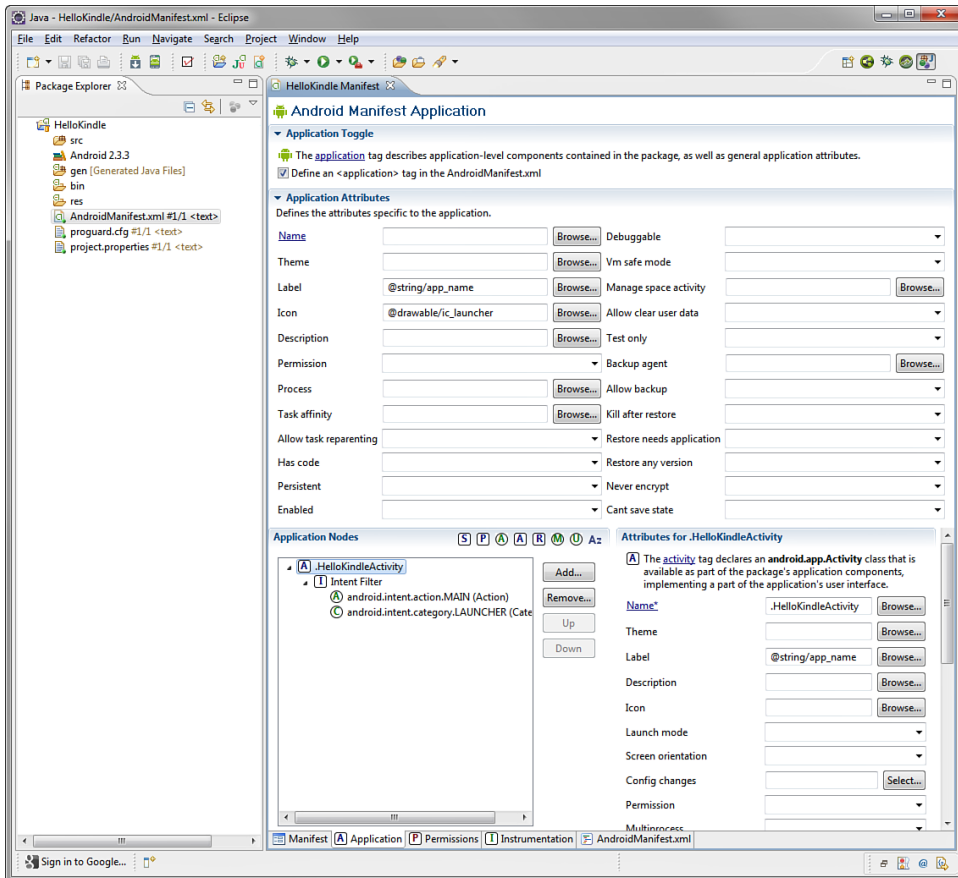


Figure 5.2 The Application Tab of the Eclipse Manifest File Resource Editor

Using the Permissions Tab

The Permissions tab (see Figure 5.3) contains any permission rules required by the application. This tab can also be used to enforce custom permissions created for the application.

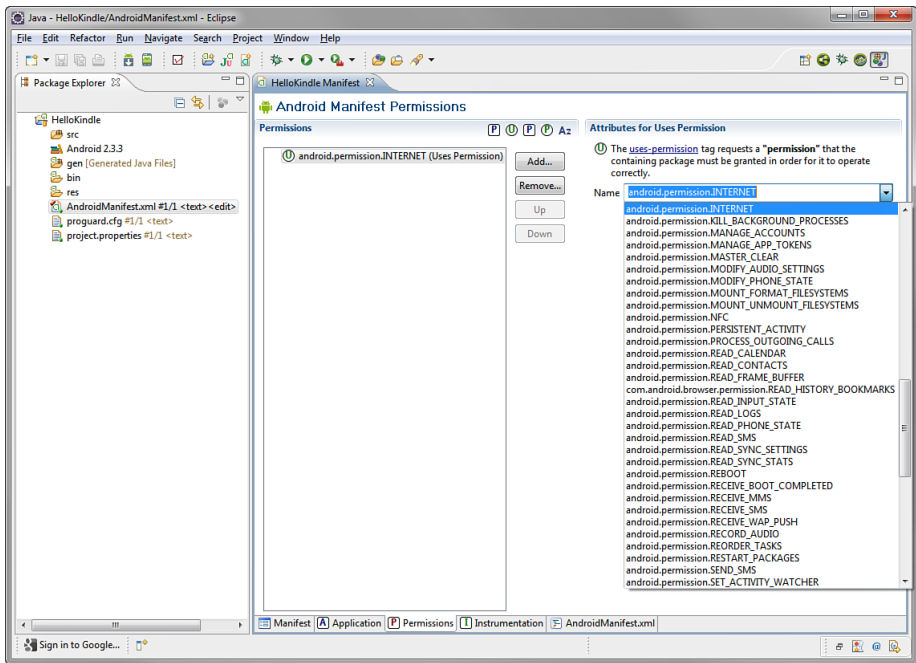


Figure 5.3 The Permissions Tab of the Eclipse Manifest File Resource Editor

Using the Instrumentation Tab

You can use the Instrumentation tab (see Figure 5.4) to declare any instrumentation classes for monitoring the application. We talk more about testing and instrumentation in Chapter 16, “Testing Kindle Fire Applications.”

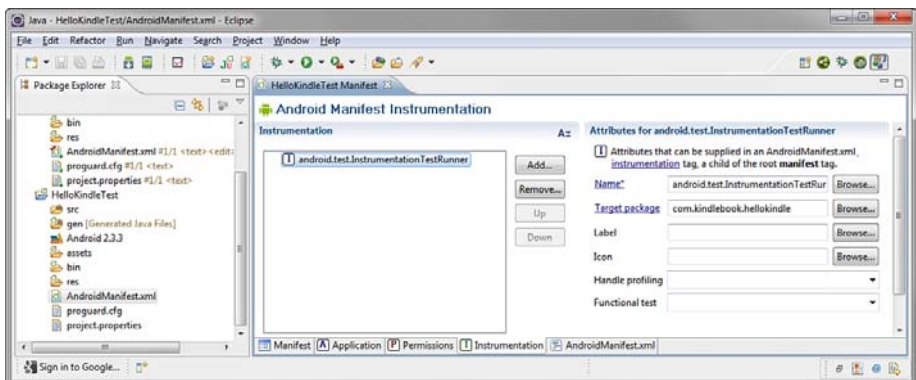


Figure 5.4 The Instrumentation Tab of the Eclipse Manifest File Resource Editor

Using the AndroidManifest.xml Tab

The Android manifest file is a specially formatted XML file. You can edit the XML manually in the AndroidManifest.xml tab of the manifest file resource editor (see Figure 5.5).

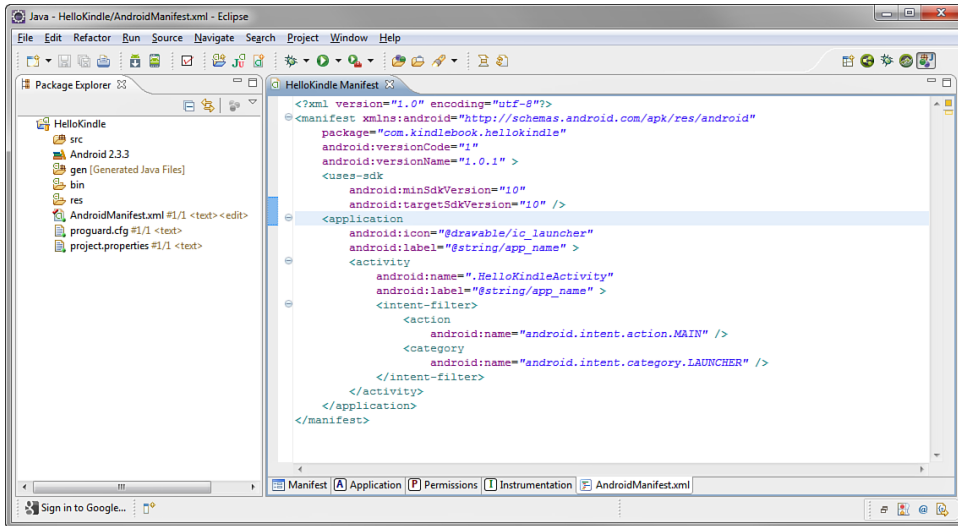


Figure 5.5 The AndroidManifest.xml Tab of the Eclipse Manifest File Resource Editor

Figure 5.5 shows the Android manifest file for the `HelloKindle` project you created in Chapter 1, “Getting Started with Kindle Fire,” which has fairly simple XML.

Note that the file has a single `<manifest>` tag, within which all the package-wide settings appear. Within this tag is one `<application>` tag, which defines the specific application, with its single activity, called `.HelloKindleActivity`, with an Intent filter. The prefixed dot (.) simply tells the system that the activity is in the package identified by the manifest package setting. In addition, the `<uses-sdk>` tag is set to target only API Level 10 (Android 2.3) for this example.

Now, let’s talk about each setting in more detail.

Configuring Basic Application Settings

If you use the Android project wizard in Eclipse to create a project, an Android manifest file will be created for you by default. However, this is just a starting point. It is important to become familiar with how the Android manifest file works; if your application’s manifest file is configured incorrectly, your application may not run properly.

In terms of the XML definition for the Android manifest file, it always starts with an XML header:

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

Many of the important settings that your application requires are set using attributes and child tags of the `<manifest>` and `<application>` blocks. Now, let's look at a few of the most common manifest file settings.

Naming Android Packages

You define the details of the application within the scope of the `<manifest>` tag. This tag has a number of essential attributes, such as the application package name. Set this value using the `package` attribute, as follows:

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidbook.hellokindle"
    android:versionCode="1"
    android:versionName="1.0.1">
```

Versioning an Application

Manifest version information is used for two purposes:

- To organize and keep track of application features
- To manage application upgrades

For this reason, the `<manifest>` tag has two separate version attributes: a version name and a version code.

Setting the Version Name

The version name is the traditional versioning information used to track application builds. Smart versioning is essential when publishing and supporting applications. The `<manifest>` tag `android:versionName` attribute is a string value provided to track the application build number. For example, the `HelloKindle` project has the version name `1.0.1`. The format of the version name field is up to the developer. However, note that this field is visible to the user.

Setting the Version Code

The version code allows the Android platform to programmatically upgrade and downgrade an application. The `<manifest>` tag `android:versionCode` attribute is a whole number integer value that the Android platform and Android marketplaces use to manage application upgrades and downgrades. `android:versionCode` generally starts at a value of 1. This value must be incremented with each update of the application deployed to users. The version code field is not visible to the user and does not need to stay in sync with the version name. For example,

an update might have a version name of 1.0.2, but the version code would be incremented to 2.

Setting the Minimum Android API Version

Android applications can be compiled for compatibility with several different SDK versions. You use the `<uses-sdk>` tag to specify the minimum SDK required in order for the application to build and run properly. The `android:minSdkVersion` attribute of this tag is an integer representing the minimum Android SDK version required. Table 5.1 shows the Android SDK versions available for shipping applications.

Table 5.1 Relevant Android SDK Versions for the Kindle Fire

Android SDK Version	Value
Android 1.0 SDK	1
Android 1.1 SDK	2
Android 1.5 SDK	3
Android 1.6 SDK	4
Android 2.0 SDK	5
Android 2.0.1 SDK	6
Android 2.1.x SDK	7
Android 2.2.x SDK	8
Android 2.3, 2.3.1, 2.3.2 SDK	9
Android 2.3.3, 2.3.4 SDK	10 (<= Kindle Fire)
Android 3.0.x SDK	11
Android 3.1.x SDK	12
Android 3.2 SDK	13
Android 4.0, 4.0.1, 4.0.2 SDK	14
Android 4.0.3	15

For example, in the `HelloKindle` project, you specified the minimum SDK as Android 2.3.3 SDK (as API Level 10, which is also 2.3.4, which is what the Kindle Fire runs):

```
<uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="10" />
```

Each time a new Android SDK is released, you can find the SDK version number in the SDK release notes. This is often referred to as the API level within the tools, especially the Android

SDK and AVD Manager. For an up-to-date list of the available API levels, see <http://goo.gl/n0fUZ>. The value need not be a number, as witnessed by the Honeycomb Preview SDK with an API Level of `HONEYCOMB`.

As of this writing, it's unclear what major SDK updates the Kindle Fire will receive. The user interface is already thoroughly customized and much different from any stock version of Android. For users, Kindle Fire doesn't necessarily need a new version of the Android platform in the future, above and beyond bug fixes.

Naming an Application

The `<application>` tag `android:label` attribute is a string representing the application name. You can set this name to a fixed string, as in the following example:

```
<application android:label="My application name">
```

You can also set the `android:label` attribute to a string resource. In the `HelloKindle` project, for example, the application name is set to the `label` string resource, as follows:

```
<application android:label="@string/app_name">
```

In this case, the resource string called `app_name` in the `strings.xml` file supplies the application name.

Providing an Icon for an Application

The `<application>` tag attribute called `android:icon` references a `Drawable` resource representing the application. There is a default icon (named `ic_launcher`) created by the new application wizard. In the `HelloKindle` project, you could set a custom application icon to the `Drawable` resource you include in the project (such as `icon.png`), as follows:

```
<application android:icon="@drawable/ic_launcher">
```

Providing an Application Description

The `<application>` tag `android:description` attribute is a string representing a short description of the application. You can set this name to a string resource:

```
<application
    android:label="My application name"
    android:description="@string/app_desc">
```

The Android system and application marketplaces use the application description to display information about the application to the user.

Setting Debug Information for an Application

The `<application>` tag `android:debuggable` attribute is a Boolean value that indicates whether the application can be debugged using a debugger such as Eclipse. This value is automatically set when you do a debug build in Eclipse, but this hasn't always been the case. If you manually turn it on or find it on in legacy applications, you must set this value to `false` before you publish your application. If you forget, the publishing tools will usually warn you to adjust this setting.

Setting Other Application Attributes

Numerous other settings appear on the Application tab, but they generally apply only in very specific cases, such as when you want to link secondary libraries or apply a theme other than the default to your application. There are also settings for handling how the application interacts with the Android operating system. For most applications, the default settings are acceptable.

You will spend a lot of time on the Application tab in the Application Nodes box, where you can register application components—most commonly, each time you register a new activity.

Defining Activities

Recall that Android applications comprise a number of different activities. Every activity must be registered within the Android manifest file by its class name before it can be run on the device. You will therefore need to update the manifest file each time you add a new activity class to an application.

Each activity represents a specific task to be completed, often with its own screen. Activities are launched in different ways using the `Intent` mechanism. Each activity can have its own label (name) and icon, but it uses the application's generic label and icon by default.

Registering Activities

You must register each activity in the Application Nodes section of the Application tab. Each activity has its own `<activity>` tag in the resulting XML. For example, the following XML excerpt defines an activity class called `HelloKindleActivity`:

```
<activity android:name=".HelloKindleActivity" />
```

This activity must be defined as a class within the application package. If needed, you may specify the entire name, including package, with the activity class name. For example, to register a new activity in the `HelloKindle` project, follow these steps:

1. Open the `HelloKindle` project in Eclipse.
2. Right-click `/src/com.kindlebook.hellokindle` and choose New, Class. The New Java Class window opens.

3. Name your new class `HelloKindleActivity2`.
4. Click the Browse button next to the Superclass field and set the superclass to `android.app.Activity`. You may need to type several letters of the class/package name before it resolves and you can choose it from the list.
5. Click the Finish button. You see the new class in your project.
6. Make a copy of the `main.xml` layout file in the `/res/layout` resource directory for your new activity and name it `second.xml`. Modify the layout so that you know it's for the second activity. For example, you could change the text string shown. Save the new layout file.
7. Open the `HelloKindleActivity2` class. Right-click within the class and choose Source, Override/Implement Methods.
8. Check the box next to the `onCreate(Bundle)` method. This method is added to your class.
9. Within the `onCreate()` method, set the layout to load for the new activity by adding and calling the `setContentView(R.layout.second)` method. Save the class file.
10. Open the Android manifest file and click the Application tab of the resource editor.
11. In the Application Nodes section of the Application tab, click the Add button and choose the Activity element. Make sure that you are adding a top-level activity. The attributes for the activity are shown on the right side of the screen.
12. Click the Browse button next to the activity Name field. Choose the new activity you created, `HelloKindleActivity2`.
13. Save the manifest file. Switch to the `AndroidManifest.xml` tab to see what the new XML looks like.

You now have a new, fully registered `HelloKindleActivity2` activity that you can use in your application.

Designating the Launch Activity

You can use an `Intent` filter to designate an activity as the primary entry point of the application. The `Intent` filter for launching an activity by default must be configured using an `<intent-filter>` tag with the `MAIN` action type and the `LAUNCHER` category. In the `HelloKindle` project, the Android project wizard sets `HelloKindleActivity` as the primary launching point of the application:

```
<activity
    android:name=".HelloKindleActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action
            android:name="android.intent.action.MAIN" />
```

```

<category
    android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

```

This `<intent-filter>` tag instructs the Android system to direct all application-launch requests to the `HelloKindleActivity` activity.

Managing Application Permissions

The Android platform is built on a Linux kernel and leverages its built-in system security as part of the Android security model. Each Android application exists in its own virtual machine and operates within its own Linux user account (see Figure 5.6).

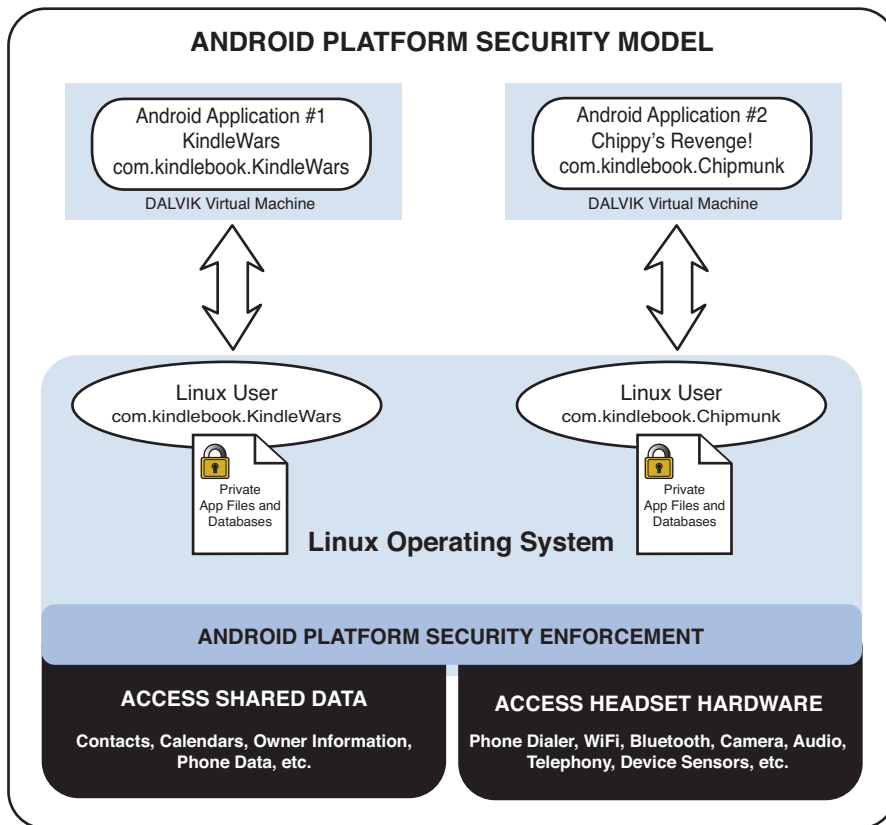


Figure 5.6 Simplified Android Platform Architecture from a Security Perspective

Applications that want access to shared or privileged resources must declare those specific permissions in the Android manifest file. This security mechanism ensures that no application can change its behavior on-the-fly or perform any operations without the user's permission.

Android applications can access their own private files and databases without any special permissions. However, if an application needs to access shared or sensitive resources, it must declare those permissions using the `<uses-permission>` tag within the Android manifest file. These permissions are managed on the Permissions tab of the Android manifest file resource editor.

For example, to give your application permission to access the network resources, follow these steps:

1. Open the `HelloKindle` project in Eclipse.
2. Open the Android manifest file and click the Permissions tab of the resource editor.
3. Click the Add button and choose Uses Permission. The Name attribute for the permission is shown on the right side of the screen as a dropdown list.
4. Choose `android.permission.INTERNET` from the dropdown list.
5. Save the manifest file. Switch to the `AndroidManifest.xml` tab to see what the new XML looks like.

You have now registered the Internet permission. Your application will be able to access all networking APIs and any non-networking APIs that can read from Internet URLs (uses the network internally) within the Android SDK without causing security exceptions to be thrown.

Table 5.2 lists some of the most common permissions used by Android applications.

Table 5.2 Common Permissions Used by Kindle Fire Applications

Permission Category	Useful Permissions
Accessing contact database	<code>android.permission.READ_CONTACTS</code> <code>android.permission.WRITE_CONTACTS</code>
Using network sockets	<code>android.permission.INTERNET</code>
Accessing audio settings	<code>android.permission.MODIFY_AUDIO_SETTINGS</code>
Accessing network settings	<code>android.permission.ACCESS_NETWORK_STATE</code> <code>android.permission.CHANGE_NETWORK_STATE</code>
Accessing Wi-Fi settings	<code>android.permission.ACCESS_WIFI_STATE</code> <code>android.permission.CHANGE_WIFI_STATE</code>
Accessing battery settings	<code>android.permission.BATTERY_STATS</code>

During the application-installation process, the user is shown exactly what permissions the application uses. The user must agree to install the application after reviewing these permissions. For a complete list of the permissions used by Android applications, see the `android.Manifest.permission` class documentation.

Managing Other Application Settings

In addition to the features already discussed in this chapter, numerous other specialized features can be configured in the Android manifest file. For example, if your application requires a hardware keyboard or a touch screen, you can specify these hardware configuration requirements in the Android manifest file.

You must also declare any other application components—such as whether your application acts as a service provider, content provider, or broadcast receiver—in the Android manifest file.

Summary

The Android manifest file (`AndroidManifest.xml`) exists at the root of every Android project. It is a required component of any application. The Android manifest file can be configured using the manifest file editor built into Eclipse by the ADT plug-in, or you can directly edit the manifest file XML. The file uses a simple XML schema to describe what the application is, what its components are, and what permissions it has. The Android platform uses this information to manage the application and grant its activities certain permissions on the Android operating system.

Exercises

1. Review the complete list of available permissions for Android applications in the Android SDK documentation. You can do this with your local copy of the documentation or online at the Android Developer website (<http://goo.gl/II3Uv>).
2. Edit the Android manifest file for the `HelloKindle` application again. Add a second permission (any will do; this is just for practice) to the application. Look up what that permission is used for in the documentation, as discussed in exercise 1.
3. Add another `Activity` class to the `HelloKindle` application and register this new `Activity` within the Android manifest. Take this exercise a step further and make this new `Activity` class your application's default launch activity with the proper intent filter. (More than one activity can be a launcher activity. Each one with the launcher category will appear in the application list with an icon. This is not typical, so you may want to move the intent filter rather than copy it.) Save your changes and run your application.

This page intentionally left blank

Designing an Application Framework

It's time to put the skills you have learned so far to use and write some code. In this chapter, you design an Android application prototype—the basic framework upon which you will build a full application. Taking an iterative approach, you will add many exciting features to this application over the course of this book. So, let's begin.

Designing an Android Trivia Game

Social trivia-style games are always popular. They are also an application category where you can, from a development perspective, explore many different features of the Android SDK. So, let's implement a fairly simple trivia game, and by doing so, learn all about designing an application user interface, working with text and graphics, and, eventually, connecting with other users.

We need a theme for our game. How about reading? In our soon-to-be-viral game, the users will be asked whether or not they've read a particular book. If they answer yes, they'll get a point. If they answer no, they'll get an opportunity to buy the book to read and improve their score.

The user with the highest score is the most well read and cultured. Let's call the game Have You Read That?.

Determining High-Level Game Features

First, we need to roughly sketch out what we want this application to do. Imagine what features a good application should have, and what features a trivia application needs. In addition to the game question screen, the application will likely need the following:

- A splash sequence that displays the app name, version, and developer
- A way to view scores

- An explanation of the game rules
- A way to store game settings

You also need a way to transition between these different features. One way to do this is to create a traditional main menu screen that the user can use to navigate throughout the application.

Reviewing these requirements, we find that we need six primary screens within the Have You Read That? application:

- Startup screen
- Main menu screen
- Game play screen
- Settings screen
- Scores screen
- Help screen

These six screens make up the core user interface for the Have You Read That? application.

Determining Activity Requirements

Each screen of the Have You Read That? application will have its own `Activity` class. Figure 6.1 shows the six activities required, one for each screen.

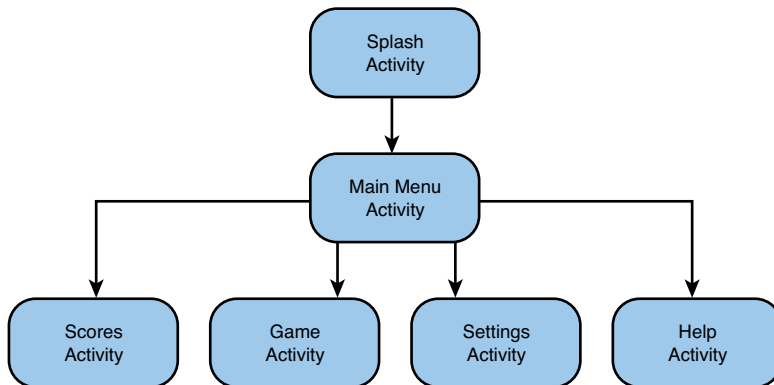


Figure 6.1 Rough Design of the Activity Workflows in the Have You Read That? Application

A good design practice is to implement a base `Activity` class with shared components, which we'll simply call `QuizActivity`. You will employ this practice as you define the activities needed by the Have You Read That? game:

- **QuizActivity**—Derived from `android.app.Activity`, this is the base class. Here, you define application preferences and other application-wide configuration and shared functionality.
- **QuizSplashActivity**—Derived from `QuizActivity`, this class represents the splash screen.
- **QuizMenuActivity**—Derived from `QuizActivity`, this class represents the main menu screen.
- **QuizHelpActivity**—Derived from `QuizActivity`, this class represents the help screen.
- **QuizScoresActivity**—Derived from `QuizActivity`, this class represents the scores screen.
- **QuizSettingsActivity**—Derived from `QuizActivity`, this class represents the settings screen.
- **QuizGameActivity**—Derived from `QuizActivity`, this class represents the game screen.

Determining Screen-Specific Game Features

Now, it's time to define the basic features of each activity in the Have You Read That? application.

Defining Splash Screen Features

The splash screen serves as the initial entry point for the Have You Read That? game. Its functionality should be encapsulated within the `QuizSplashActivity` class. This screen should do the following:

- Display the name and version of the application
- Display an interesting graphic or logo for the game
- Transition automatically to the main menu screen after a period of time

Figure 6.2 shows a hand-drawn mockup of the splash screen.

Defining Main Menu Screen Features

The main menu screen serves as the main navigational screen in the game. This screen displays after the splash screen and requires the user to choose where to go next. Its functionality should be encapsulated within the `QuizMenuActivity` class. This screen should do the following:

- Automatically display after the splash screen
- Allow the user to choose Play Game, Settings, Scores, or Help

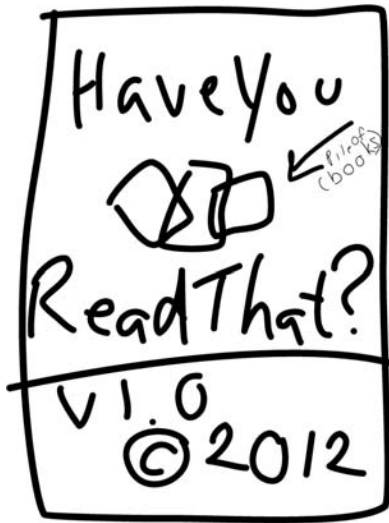


Figure 6.2 The Have You Read That? Splash Screen

Figure 6.3 shows a hand-drawn mockup of the main menu screen.



Figure 6.3 The Have You Read That? Main Menu Screen

Defining Help Screen Features

The help screen tells the user how to play the game. Its functionality should be encapsulated within the `QuizHelpActivity` class. This screen should do the following:

- Display help text to the user and enable the user to scroll through text
- Provide a method for the user to suggest new questions

Figure 6.4 shows a hand-drawn mockup of the help screen.

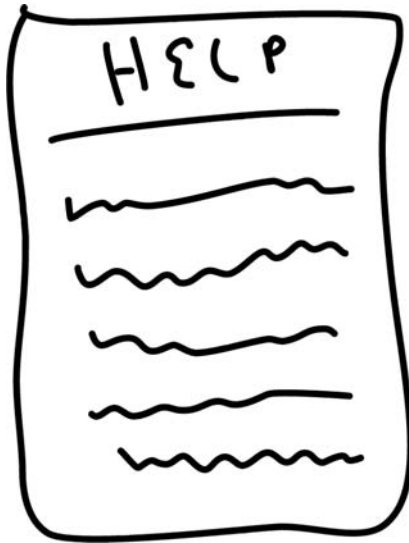


Figure 6.4 The Have You Read That? Help Screen

Defining Scores Screen Features

The scores screen allows the user to view game scores. Its functionality should be encapsulated within the `QuizScoresActivity` class. This screen should do the following:

- Display top score statistics
- Show the latest score if the user is coming from the game screen

Figure 6.5 shows a hand-drawn mockup of the scores screen.



Scores		
All Friends		
Name	Score	Rank
LED	100	1
SAC	60	2
ESC	1	3

Figure 6.5 The Have You Read That? Scores Screen

Defining Settings Screen Features

The settings screen allows users to edit and save game settings, including username and other important features. Its functionality should be encapsulated within the `QuizSettingsActivity` class. This screen should do the following:

- Allow the user to input game settings
- Allow the user to invite friends to play

Figure 6.6 shows a hand-drawn mockup of the basic settings screen.

Defining Game Screen Features

The game screen displays the trivia quiz. Its functionality should be encapsulated within the `QuizGameActivity` class. This screen should do the following:

- Display a series of yes/no questions
- Handle input and keep the score and state of the quiz
- Transition to the scores screen when the user is finished playing



Figure 6.6 The Have You Read That? Settings Screen

Figure 6.7 shows a hand-drawn mockup of the game screen.



Figure 6.7 The Have You Read That? Game Screen

Implementing an Application Prototype

Now that you have a rough idea what the Have You Read That? application will do and how it will look, it's time to start coding. This involves the following steps:

1. Creating a new Android project in Eclipse
2. Adding some application resources, including strings and graphics
3. Creating a layout resource for each screen
4. Implementing a Java class (derived from the `Activity` class) for each screen
5. Creating a set of application-wide preferences for use in all activities

Reviewing the Accompanying Source Code

Because of length limitations and other practical reasons, we cannot provide full code listings in every chapter of this book—they would take more than a chapter to list and be incredibly repetitive. Instead, we provide inline code excerpts based on the Android topic at hand and provide the complete Java source code project for each chapter (the chapter is denoted by the project name, package name, and application icon) online at the publisher's website (www.informit.com/title/9780321833976) and the authors' website (<http://goo.gl/fYC7v>).

These source files are not meant to be the “answers” to a test. The full source code is vital for providing context and complete implementations of the topics discussed in each chapter of this book. We expect readers to follow along with the source code for a given chapter and, if they feel inclined, they can build their own incarnation of the Have You Read That? application in parallel. The full source code helps give context to developers less familiar with Java or mobile topics. Also, there may be times when the source code does not exactly match the code provided in this book—this is normally because we strip many comments, error checking, and exception handling from book code, again for readability and length.

For example, for Chapter 6 code, the source code Eclipse project name is `HYRT_Chapter6`, with a package name of `com.kindlebook.hyrt.chapter6` and an icon that clearly indicates the chapter number (6). This allows you to keep multiple projects in Eclipse and install multiple applications on a single device without conflicts or naming clashes. However, if you are building your own version in parallel, you likely would only have one version—a single Eclipse project, one application you revise and improve in each chapter—using the downloaded project for reference.

Creating a New Android Project

You can begin creating a new Android project for your application by using the Eclipse Android project wizard.

The project has the following settings:

- **Project name**—HYRT (Note: For this chapter's source code, this chapter's project is named `HYRT_Hour6`.)
- **Build target**—API Level 10 (Android Open Source Project as vendor)
- **Application name**—Have You Read That?
- **Package name**—`com.kindlebook.hyrt` (Note: For this chapter's source code, the package is actually named `com.kindlebook.hyrt.chapter6`.)
- **Create activity**—`QuizSplashActivity`

Using these settings, you can create the basic Android project. However, you need to make a few adjustments.

Adding Project Resources

The Have You Read That? project requires some additional resources. Specifically, you need to add a `Layout` file for each activity and a text string for each activity name, and you need to change the application icon to something more appropriate.

Adding String Resources

Begin by modifying the `strings.xml` resource file. Delete the `hello` string and create six new string resources—one for each screen. For example, create a string called `help` with a value of `Help Screen`. When you are done, the `strings.xml` file should look similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string
        name="app_name">Have You Read That?</string>
    <string
        name="help">Help Screen</string>
    <string
        name="menu">Main Menu Screen</string>
    <string
        name="splash">Splash Screen</string>
    <string
        name="settings">Settings Screen</string>
    <string
        name="game">Game Screen</string>
    <string
        name="scores">Scores Screen</string>
</resources>
```


Adding Layout Resources

Next, you need layout resource files for each activity. Begin by renaming the `main.xml` layout to `splash.xml`. Then, copy the `splash.xml` file five more times, resulting in one layout for each activity: `game.xml`, `help.xml`, `menu.xml`, `scores.xml`, and `settings.xml`.

You may notice that there is an error in each Layout file. This is because the `TextView` control in the layout refers to the `@string/hello` string, which no longer exists. For each layout file, you need to use the Eclipse layout editor to change the `String` resource loaded by the `TextView` control. For example, `game.xml` needs to replace the reference to `@string/hello` with the new string you created called `@string/game`. Now when each layout loads, it displays the screen that it is supposed to represent.

Adding Drawable Resources

While you are adding resources, you should change the icon for your application to something more appropriate. To do this, create a 48×48 pixel PNG file called `ic_launcher.png` (ic for icon, launcher for the launcher screen), and add this resource file to the `/drawable-mdpi` resource directory. This replaces the default `ic_launcher.png` file.

For the Kindle Fire source code, only the `/drawable-mdpi` directory matches the density of the Kindle Fire screen. However, even if you've created four differently sized icons to support different types of device screens and placed them in the four main drawable resource directories (`/drawable-ldpi`, `/drawable-mdpi`, `/drawable-hdpi`, and `/drawable-xhdpi`), only a single reference to the icon is required. Just make sure that all the icons are named identically. This enables the Android operating system to choose the most appropriate icon version for the device. Note that if you don't create the other three icons because you're only targeting the Kindle Fire, Android Lint will give warnings about the missing icons.

Launcher Icons on Kindle Fire

Applications displayed on the Kindle Fire home screen do not use standard launcher icons. Instead, the large icons come from an Amazon web service tied to the images you upload to Amazon Appstore. When the application is not installed by downloading it via Amazon's Appstore, these images are not available. In this scenario, the home screen uses the launcher icon supplied in the application package, as described here.

Implementing Application Activities

To implement a base `Activity` class, simply copy the source file called `QuizSplashActivity.java`. Name this new class file `QuizActivity` and save the file. This class should look simple for now:

```
package com.kindlebook.hyrt;

import android.app.Activity;
```

```
public class QuizActivity extends Activity {  
    public static final String GAME_PREFERENCES = "GamePrefs";  
}
```

You add to this class later. Next, update the `QuizSplashActivity` class to extend from the `QuizActivity` class instead of directly from the `Activity` class.

Creating the Rest of the Application Activities

Now, perform the same steps five more times, once for each new activity: `QuizMenuActivity`, `QuizHelpActivity`, `QuizScoresActivity`, `QuizSettingsActivity`, and `QuizGameActivity`. Note the handy way that Eclipse updates the class name when you copy a class file. You can also create class files by right-clicking the package name `com.kindlebook.hyrt` and choosing **New Class**. Eclipse presents a dialog where you can fill in class file settings.

Note that there is an error in each Java file. This is because each activity is trying to load the `main.xml` layout file—a resource that no longer exists. You need to modify each class to load the specific layout associated with that activity. For example, in the `QuizHelpActivity` class, modify the `setContentView()` method to load the layout file you created for the help screen, as follows:

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

You need to make similar changes to the other activity files, such that each call to `setContentView()` loads the corresponding layout file. For more tips on working with Eclipse, check out Appendix B, “Eclipse IDE Tips and Tricks.”

Updating the Android Manifest File

You now need to make some changes to the Android manifest file. First, modify the application icon resource to point at the `@drawable/ic_launcher` icon you created, if it doesn’t already. Second, you need to register all of your new activities in the manifest file so they will run properly. Finally, verify that you have `QuizSplashActivity` set as the default activity to launch.

Creating Application Preferences

The Have You Read That? application needs a simple way to store some basic state information and user data. You can use Android’s shared preferences (`android.content.SharedPreferences`) to add this functionality.

You can access shared preferences, by name, from any activity within the application. Therefore, declare the name of your set of preferences in the base class `QuizActivity` so that they are easily accessible to all subclasses:

```
public static final String GAME_PREFERENCES = "GamePrefs";
```

Here are the steps you need to take to save shared preferences:

1. Use the `getSharedPreferences()` method to retrieve an instance of a `SharedPreferences` object within your `Activity` class.
2. Create a `SharedPreferences.Editor` object to modify preferences.
3. Make changes to the preferences by using the editor.
4. Commit the changes by using the `commit()` method in the editor.

You don't need to do this now; we'll add preferences when we need them.

Saving Specific Shared Preferences

Each preference is stored as a key/value pair. Preference values can be the following types:

- Boolean
- Float
- Integer
- Long
- String

After you decide what preferences you want to save, you need to get an instance of the `SharedPreferences` object and use the `Editor` object to make the changes and commit them. The following sample code, when placed within an `Activity` class, illustrates how to save two preferences—the user's name and age:

```
import android.content.SharedPreferences;
// ...
SharedPreferences settings =
    getSharedPreferences(GAME_PREFERENCES, MODE_PRIVATE);
SharedPreferences.Editor prefEditor = settings.edit();
prefEditor.putString("UserName", "JaneDoe");
prefEditor.putInt("UserAge", 22);
prefEditor.commit();
```

You can also use the shared preferences editor to clear all preferences, using the `clear()` method, and to remove specific preferences by name, using the `remove()` method.

Retrieving Shared Preferences

Retrieving shared preference values is even simpler than creating them because you don't need an editor. The following example shows how to retrieve shared preference values within your `Activity` class:

```
SharedPreferences settings =  
    getSharedPreferences(GAME_PREFERENCES, MODE_PRIVATE);  
if (settings.contains("UserName") == true) {  
    // We have a username  
    String user = Settings.getString("UserName", "Default");  
}
```

You can use the `SharedPreferences` object to check for a preference by name, retrieve strongly typed preferences, or retrieve all the preferences and store them in a map.

Although you have no immediate needs for shared preferences yet in *Have You Read That?*, you now have the infrastructure set up to use them as needed within any of the activities within your application. This will be important later when you implement each activity in full in subsequent chapters.

Running the Game Prototype

You are almost ready to run and test your application. But first, you need to create a debug configuration for your new project within Eclipse.

Creating a Debug Configuration

Each new Eclipse project requires a debug configuration. Be sure to set the preferred AVD for the project to one that is compatible with the Google APIs and within the API level target range you set in your application. (Check your Android Manifest file if you are unsure.) If you do not have one configured appropriately, simply click the Android SDK and AVD Manager button in Eclipse. From here, determine which AVDs are appropriate for the application and create new ones, as necessary.

Launching the Prototype in the Emulator

It's time to launch the *Have You Read That?* application in the Android emulator. You can do this by using the little bug icon in Eclipse or by clicking the Run button on the debug configuration you just created.

As you see in Figure 6.8, the application does very little so far. It has a pretty icon, which a user can click to launch the default activity, `QuizSplashActivity`. This activity displays its `TextView` control, informing you that you have reached the splash screen. There is no real user interface to speak of yet for the application, and you still need to wire up the transitions between the different activities. However, you now have a solid framework upon which to build. In the next few chapters, you flesh out the different screens and begin to implement game functionality.

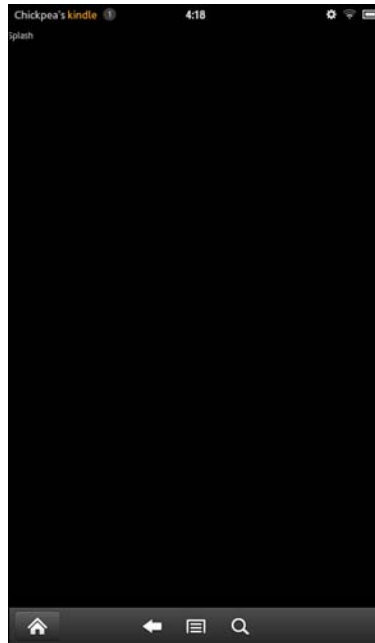


Figure 6.8 The Splash Screen for Have You Read That?

Exploring the Prototype Installation

The Have You Read That? application does very little so far, but you can use tools on the Android emulator to peek at all you've done so far:

- **Application Manager**—This is helpful for determining interesting information about an application. In the emulator, navigate to the home screen, click the Menu button and choose Settings, Applications, Manage applications, and then choose the Have You Read That? application from the list of applications. Here, you can see some basic information about the application, including storage and permissions used, as well as information about the cache and so on. You can also kill the app or uninstall it.
- **Dev Tools**—This tool helps you inspect the application in more detail. In the emulator, pull up the application drawer, launch the Dev Tools application, and choose Package Browser. Navigate to the package name `com.kindlebook.hyrt`. This tool reads information out of the manifest and allows you to inspect the settings of each activity registered, among other features.

Of course, you can also begin to investigate the application by using the DDMS perspective of Eclipse. For example, you could check out the application directory for the `com.kindlebook.hyrt` package on the Android file system. You could also step through the code of `QuizSplashActivity`.

Summary

In this chapter, you built a basic prototype on which you can build in subsequent chapters. You designed a prototype and defined its requirements in some detail. Then, you created a new Android project, configured it, and created an activity for each screen. You also added custom layouts and implemented shared preferences for the application.

Exercises

1. Add a log message to the `onCreate()` method of each Activity class in your Have You Read That? application prototype. For example, add an informational log message, such as `In Activity QuizSplashActivity` to the `QuizSplashActivity` class.
2. Add an additional application preference string to the application prototype: `lastLaunch`. In the `onCreate()` method of the `QuizSplashActivity` class, make the following changes. Whenever this method runs, read the old value of the `lastLaunch` preference and print its value to the log output. Then, update the preference with the current date and time.

Hint: The default `Date` class (`java.util.Date`) constructor can be used to get the current date and time, and the `SimpleDateFormat` class (`java.text.SimpleDateFormat`) can be used to format date and time information in various string formats. See the Android SDK for complete details on these classes.

3. Sketch out an alternate design for the Have You Read That? application. Consider options such as not including the Main Menu screen. Look over similar applications in the Android Market for inspiration. You can post links to alternative designs for the application on this book's website (<http://goo.gl/gPguA>) or email them directly to us at androidwirelessdev+hyrt@gmail.com.

This page intentionally left blank



Building an Application Framework

- 7** Implementing an Animated Splash Screen 101
- 8** Implementing the Main Menu Screen 117
- 9** Developing the Help and Scores Screens 133
- 10** Collecting User Input 149
- 11** Using Dialogs to Collect User Input 165
- 12** Adding Application Logic 181
- 13** Adding Network Support 199
- 14** Exploring the Amazon Web Services SDK for Android 225

This page intentionally left blank

Implementing an Animated Splash Screen

In this chapter, we focus on implementing the splash screen of the Have You Read That? application. After roughly sketching out the screen design, you determine exactly which Android view controls you need to implement the `splash.xml` layout file. When you are satisfied with the screen layout, you add some tweened animations to give the splash screen some pizzazz. Finally, after your animations are complete, you must implement a smooth transition from the splash screen to the main menu screen.

Designing the Splash Screen

You'll implement the Have You Read That? application from the ground up, beginning with the screen users see first: the splash screen. Recall from Chapter 6, "Designing an Application Framework," that you had several requirements for this screen. Specifically, the screen should display some information about the application (title and version information) in a visually appealing way and then, after some short period of time, automatically transition to the main menu screen. Figure 7.1 provides a rough design for the splash screen.

For the time being, you focus on designing the splash screen in portrait mode, but you will try to avoid making the porting effort difficult for landscape or square orientations. For now, this simple layout design will suffice. Different devices will display this layout in different ways. We discuss porting issues and how to support different devices later in this book.

Recall that the full source code associated with this chapter is available on this book's websites.

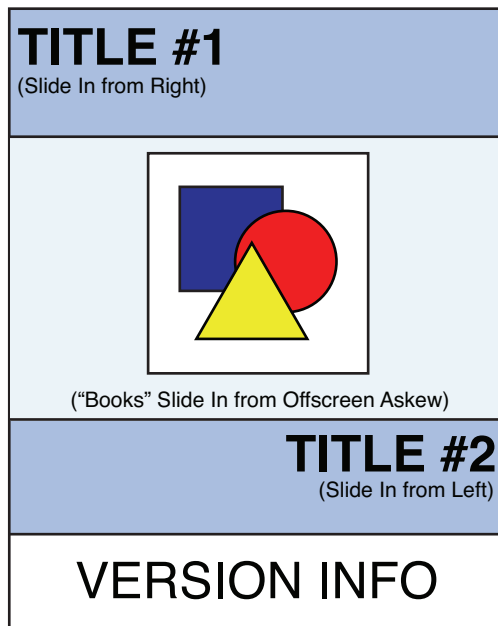


Figure 7.1 Rough Design for the Have You Read That? Splash Screen

Implementing the Splash Screen Layout

Now that you know how your splash screen should look, you need to translate the rough design into the appropriate layout design. Recall that the `/res/layout/splash.xml` layout file is used by `QuizSplashActivity`. You need to update the default layout, which simply displays a single `TextView` control (informing us it is the splash screen) to contain controls for each of the elements in the rough design.

Screen view controls come in many forms. Each control is a rectangle that can control a specific part of the screen. You are using two common screen controls on your splash screen:

- A `TextView` control displays a text string onscreen.
- An `ImageView` control displays a graphic onscreen.

You also need some way to organize various view controls on the screen in an orderly fashion. For this, you use `Layout` controls. You've seen `LinearLayout` in the default files, which allows placement of child views that are vertically or horizontally aligned.

In addition to `LinearLayout`, there are a number of other `Layout` controls. Layouts may be nested and control only part of the screen, or they may control the entire screen. It is common for a screen to be encapsulated in one large parent layout—often a `LinearLayout` control. Table 7.1 lists the available `Layout` controls.

Table 7.1 Common Layout Controls

Layout Control Name	Description	Key Attributes/Elements
<code>LinearLayout</code>	Each child view is placed after the previous one, in a single row or column.	Orientation (vertical or horizontal).
<code>RelativeLayout</code>	Each child view is placed in relation to the other views in the layout or relative to the edges of the parent layout.	Many alignment attributes to control where a child view is positioned relative to other child view controls.
<code>FrameLayout</code>	Each child view is stacked within the frame, relative to the top-left corner. view controls may overlap.	The order of placement of child view controls is important, when used with appropriate gravity settings.
<code>TableLayout</code>	Each child view is a cell in a grid of rows and columns.	Each row requires a <code>TableRow</code> element.

Layouts and their child view controls have certain attributes that help control their behavior. For example, all layouts share the attributes `android:layout_width` and `android:layout_height`, which control how wide and high an item is within a parent layout. These attribute values can be dimensions, such as a number of pixels, or use a more flexible approach: `match_parent` or `wrap_content`. Using `match_parent` instructs a layout to scale to the size of the parent layout, and using `wrap_content` “shrink wraps” the child view control within the parent, giving it only the space of the child view control’s dimensions. A number of other interesting properties can be used to control specific layout behavior, including margin settings and type-specific layout attributes.

Let’s use a `RelativeLayout` control to display text and images on the splash screen. In the splash screen design, there are four main elements to organize; in order, these are a `TextView` control, a `FrameLayout` control with some `ImageView` controls, and then two more `TextView` controls. Figure 7.2 shows the layout design of the splash screen.

Adding New Project Resources

Now that you have your layout design for the splash screen, you need to create the string, color, and dimension resources to use within the layout.

Begin by adding five graphic resources to the `/res/drawable` directories. Specifically, you must add the following files: `splash1.png`, `splash2.png`, `splash3.png`, `splash4.png`, and a background file, `background_paper.png`, to an appropriate drawable directory. If you’re just targeting Kindle Fire, the `/mdpi` directory might be appropriate. However, if you’re also supporting smaller or larger screens with the same pixel density, but different actual screen resolutions, this may not work well. Another resource directory qualifier that can be used is `large`—for

example, `/res/drawable-large-mdpi` would more specifically identify traits matching the Kindle Fire display, allowing more flexibility in layout sizing and design.

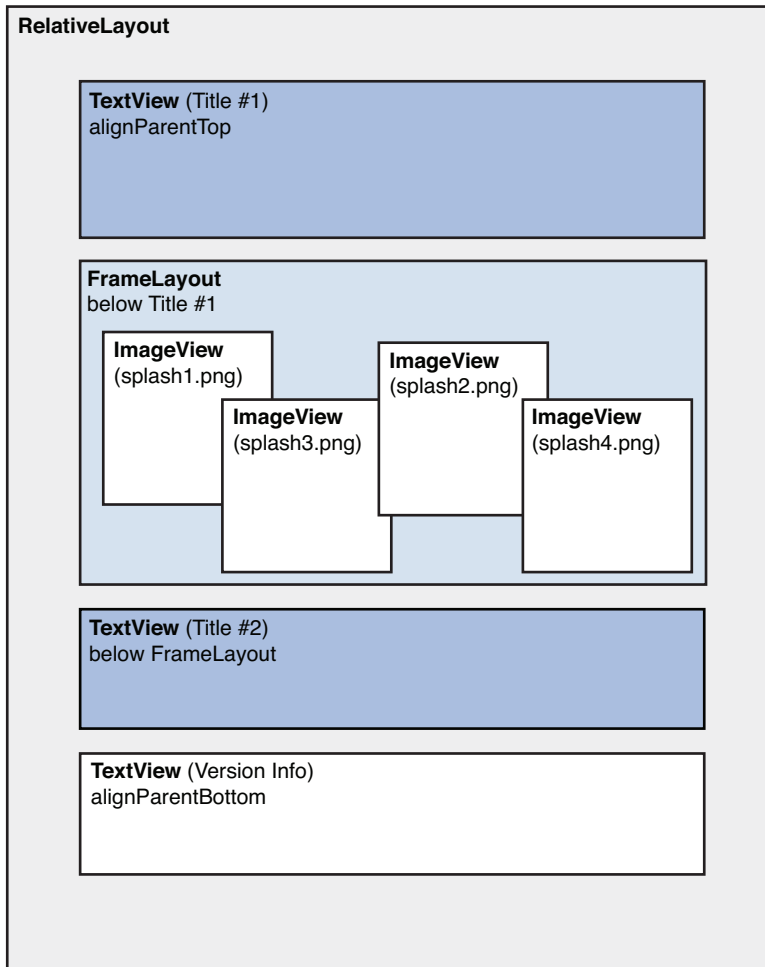


Figure 7.2 Layout Design for the Have You Read That? Splash Screen

Organizing resources with the application package is a large and complex topic—not unlike trying to explain to a computer user how to efficiently store files on the computer hard drive. Throughout this book, we highlight small examples of where Kindle Fire resources might be best put when used in an app that supports many devices. However, to keep things simple, the example we’re building in this book targets only Kindle Fire devices.

Figure 7.3 shows what the directory structure might look like within the Eclipse project.

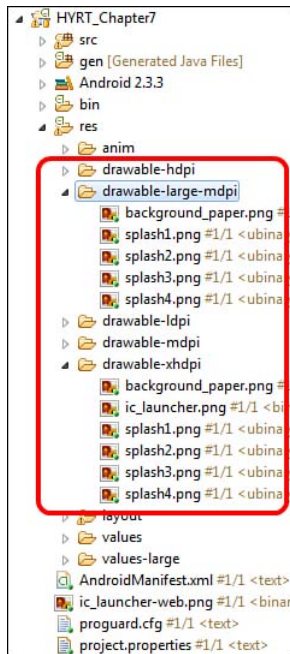


Figure 7.3 The Resource Directory Hierarchy of the Have You Read That? Application

Then, add three new strings to the `/res/values/strings.xml` resource file: one for the top title (Have You), one for the bottom title (Read That?), and one for some version information (multiple lines). Remove the `splash` string because you will no longer be using it. Your string resource file should now look similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Have You Read That?</string>
    <string name="help">Help</string>
    <string name="menu">@string/app_name</string>
    <string name="settings">Settings</string>
    <string name="game">@string/app_name</string>
    <string name="scores">Scores</string>
    <string name="splash_title_top">HAVE YOU</string>
    <string name="splash_title_bottom">READ THAT?</string>
    <string
name="app_version_info">
        Version 1.0.1\nCopyright © 2012 Mamlambo\nAll Rights Reserved.</string>
</resources>
```

Next, create a new resource file called `/res/values/colors.xml` to contain the three color resources you need: one for the title text color (dark brown), one for the version text color (grayish white), and one for the version text background color (translucent blue). Your color resource file should now look like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="splash_title_color">#412305</color>
    <color name="splash_version_color">#d0d0d0</color>
    <color name="splash_version_background">#991a1a48</color>
</resources>
```

Finally, you need to create some dimension resources in a new resource file called `/res/values/dimens.xml`. (Kindle Fire values might best be stored in a `/res/values-large/dimens.xml` file.) Create three new dimension values: one to control the title font size (75dp), one to control the version text font size (30dp), and one to allow for nice line spacing between the lines of the version text (5dp). We use the dp units so that the dimensions are flexible, device-independent values and therefore appropriate for many different resolution devices. In text displayed on the screen, such as that found on the help screen, a more appropriate dimension unit type might be sp, which scales with the user text size settings. Your dimension resource file should now look like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="splash_title_size">75dp</dimen>
    <dimen name="splash_version_size">30dp</dimen>
    <dimen name="splash_version_spacing">5dp</dimen>
</resources>
```

Save the resource files now. Once saved, you can begin to use your new resources in the `splash.xml` layout resource file.

Updating the Splash Screen Layout

Before taking the following steps, first use the editor to remove all existing controls from the `splash.xml` layout. The file should be empty except for the XML header. You can delete unwanted controls in the Graphical Layout tab by right-clicking them and choosing Delete from either the visual view or the Outline tab. However, we find that the simplest way is to delete the controls from the XML mode. After you remove any unnecessary controls, take the following steps to generate the desired layout, based on our intended design. (These steps may seem overwhelming at first, but they're important for seeing how to build up a layout. The resulting XML is shown after the steps.)

1. Begin by adding a `RelativeLayout` control and setting its background attribute to `@drawable/background_paper`. All subsequent controls will be added as child views inside this control.

2. Add a `TextView` control called `splashTitleTop`. Set `layout_width` and `layout_height` to `wrap_content`. Set the `layout_alignParentTop` value to `true`. Set the control's text attribute to the appropriate string resource, its `textColor` attribute to the appropriate color resource, its `textStyle` to `bold`, and its `textSize` to the dimension resource you created for that purpose.
3. Add a `FrameLayout` control called `splashImages`. Set its `layout_width` attribute to `match_parent` and its `layout_height` attribute to `580dp`. Set `layout_below` to `@+id/splashTitleTop` and `layout_centerHorizontal` to `true`.
4. Within the `FrameLayout` control, add four `ImageView` controls. Set each `layout_width` and `layout_height` to `wrap_content`. For the first `ImageView` control, set the `src` attribute to the `splash1.png` drawable resource called `@drawable/splash1`. Repeat for `splash2.png`, `splash3.png`, and `splash4.png`. Set each of their `layout_gravity` values to `center`. Add a `contentDescription` attribute for each. Use a string that you feel is appropriate for screen readers. This description is used by screen readers for accessibility purposes.
5. Add another `TextView` control called `splashTitleBottom` within the parent `RelativeLayout`. Set its `layout_width` attribute to `wrap_content` and `layout_height` to `wrap_content`. Set its `layout_alignParentRight` to `true` and `layout_below` to `@+id/splashImages`. Set its text attribute to the appropriate string, its `textColor` attribute to the appropriate color resource, its `textStyle` to `bold`, and its `textSize` attribute to the dimension resource you created for that purpose.
6. For the version information, create one last `TextView` control, called `splashVersion`. Set its `layout_width` attribute to `match_parent` and `layout_height` to `wrap_content`. Set its `layout_alignParentBottom` to `true`. Set its text attribute to the appropriate string, its `textColor` attribute to the appropriate color resource, its `gravity` to `center`, and its `textSize` attribute to the dimension resource you created. Also, set its `background` attribute to the color resource (dark blue).

The resulting `splash.xml` layout resource should now look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background_paper" >
    <TextView
        android:id="@+id/splashTitleTop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:text="@string/splash_title_top"
        android:textColor="@color/splash_title_color"
        android:textSize="@dimen/splash_title_size"
        android:textStyle="bold" />
```



```

<FrameLayout
    android:id="@+id/splashImages"
    android:layout_width="match_parent"
    android:layout_height="580dp"
    android:layout_below="@+id/splashTitleTop"
    android:layout_centerHorizontal="true" >
    <ImageView
        android:id="@+id/splash1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:contentDescription="@string/splash_1_desc"
        android:src="@drawable/splash1" />
    <ImageView
        android:id="@+id/splash2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:contentDescription="@string/splash_2_desc"
        android:src="@drawable/splash2" />
    <ImageView
        android:id="@+id/splash3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:contentDescription="@string/splash_3_desc"
        android:src="@drawable/splash3" />
    <ImageView
        android:id="@+id/splash4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:contentDescription="@string/splash_4_desc"
        android:src="@drawable/splash4" />
</FrameLayout>
<TextView
    android:id="@+id/splashTitleBottom"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/splashImages"
    android:text="@string/splash_title_bottom"
    android:textColor="@color/splash_title_color"
    android:textSize="@dimen/splash_title_size"
    android:textStyle="bold" />

```

```

<TextView
    android:id="@+id/splashVersion"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:background="@color/splash_version_background"
    android:gravity="center"
    android:text="@string/app_version_info"
    android:textColor="@color/splash_version_color"
    android:textSize="@dimen/splash_version_size" >
</TextView>
</RelativeLayout>

```

At this point, save all edited files and run the Have You Read That? application in the Android emulator or on your Kindle Fire. The splash screen should look like what's shown in Figure 7.4.

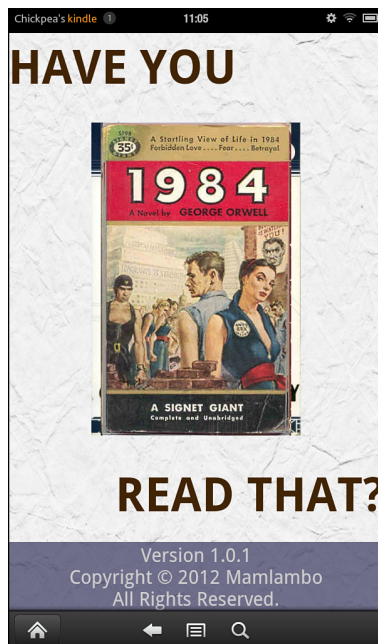


Figure 7.4 The Have You Read That? Splash Screen

Do not be concerned with the book cover images being stacked over each other. In the next section, when we discuss animation, we animate the book cover images and the intention will become as defined originally—a stack of books, askew.

Working with Animation

One great way to add zing to your splash screen is to add some animation. The Android platform supports five types of graphics animation:

- **Animated GIF images**—Animated GIFs are self-contained graphics files with multiple frames.
- **Frame-by-frame animation**—The Android SDK provides a similar mechanism for frame-by-frame animation in which the developer supplies the individual graphic frames and transitions between them (see the `AnimationDrawable` class). This is referred to as drawable animation.
- **View animation**—View animation is a simple and flexible method of defining specific animation operations that can then be applied to any view or layout. Underlying characteristics are not changed; the results are only visual. This is also referred to as tween animation.
- **Property animation**—Similar to, but far more flexible than view animation, any property on a `View` object can be transformed over time. Unlike view animation, property animation results in actual changes to the `View` objects.
- **Advanced graphics**—Android's OpenGL ES API, RenderScript API, and Canvas APIs provide advanced two-dimensional and three-dimensional drawing, animation, lighting, and texturing capabilities.

For the Have You Read That? splash screen, view animation makes the most sense. Android provides tweening support for alpha (transparency), rotation, scaling, and translating (moving) animations. You can create sets of animation operations to be performed simultaneously, in a timed sequence, and after a delay. Purely visual results are also acceptable, as the splash screen is noninteractive. Thus, view animation is a perfect choice for your splash screen.

With view animation, you create an animation sequence, either programmatically or by creating animation resources in the `/res/anim` directory. Each animation sequence needs its own XML file, but the same animation may be applied to any number of `View` controls within your application. You can also take advantage of built-in animation resources provided in the `android.R.anim` class.

Adding Animation Resources

For your splash screen, you need to create two custom animations in XML and save them to the `/res/anim` resource directory: `slide_in_1.xml` and `slide_in_2.xml`.

The first animation, `slide_in_1.xml`, translates (changes location) the image from 600 pixels to the right of its final position to its final position (as set by the layout file) over the course of 1000 milliseconds, or 1.0 seconds. There is no built-in animation editor in Eclipse. Instead, it's up to the developer to create the appropriate XML animation sequence.

An animation resource looks much like the other types of resources available. The `slide_in_1.xml` resource file simply has a single animation applied using the `<translate>` tag. It also uses an interpolator to vary the rate at which the image moves. There are several built-in interpolators. We chose the overshoot interpolator, which means the target value is overshoot before settling to the final position. For complete details on the tags and attributes available for animation resources, revisit Chapter 4, “Managing Application Resources,” or see the Android Developer online reference on the topic at <http://goo.gl/K3aZ7>.

The XML for the `slide_in_1.xml` animation should look something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/overshoot_interpolator" >
    <translate
        android:duration="1000"
        android:fromXDelta="600"
        android:toXDelta="0" />
</set>
```

You can apply this animation to the top `TextView` control with your title text.

Next, create the `slide_in_2.xml` animation. This animation does almost the same thing as the `slide_in_1` animation, except the translation starts from the left and the `startOffset` attribute should be set to 1000 milliseconds. This means that this animation will actually take 2 seconds total: It waits for 1.0 seconds and then slides in for 1.0 seconds. Because 2 seconds is long enough to display the entire splash screen, you should plan to listen for `slide_in_2` to complete and then react by transitioning to the main menu screen (more on this in a few moments).

Finally, you need to create a fun animation sequence for the `FrameLayout` graphics. Our ideal ending point is a stack of four book-cover images skewed (rotated and translated) to slightly different positions from each other in a pleasant stack. Because there is no random function available in the declarative form of defining the animations, we define them straight from Java. Performing the animation operations programmatically is not very different from using XML. Let's look at the code first:

```
private AnimationSet createSplashAnim() {
    float targetX = (float) ((Math.random()*300)-150);
    float startDegrees = (float)(Math.random() * 720)-360;
    float endDegrees = (float)(Math.random() *40)-20;

    AnimationSet tossIn = new AnimationSet(false);
    TranslateAnimation translateIn = new TranslateAnimation(600, targetX, 0, 0);
    translateIn.setInterpolator(new DecelerateInterpolator(1.0f));
```

```

    RotateAnimation rotate
        = new RotateAnimation(startDegrees, endDegrees,
            RotateAnimation.RELATIVE_TO_SELF, 0.5f,
            RotateAnimation.RELATIVE_TO_SELF, 0.5f);
    tossIn.addAnimation(translateIn);
    tossIn.addAnimation(rotate);
    tossIn.setDuration(1500);
    tossIn.setFillAfter(true);
    return tossIn;
}

```

The first three lines randomize the starting and ending positions. The images will all end up to the left or right of center by up to 150 pixels and rotated clockwise or counterclockwise by 20 degrees. Next, the `AnimationSet` object is created (just like the `<set>` tag). We add an interpolator to it, `DecelerateInterpolator`, which slows down the animation near the end. The `TranslateAnimation` object is instantiated with the final target value and a starting value that is off the screen. The `RotateAnimation` object is instantiated with the random values. These animations are added to the set, and the duration is set to 1,500 milliseconds so the animation ends right before the bottom title part ends. Finally, the `setFillAfter()` method is called with `true` so that the images stay at the animation target locations rather than returning to the original layout positions. This method should be part of the `QuizSplashActivity` class.

Now you can begin to apply the animations to specific views.

Animating Specific Views

View animation sequences must be applied and managed programmatically within your `Activity` class—in this case, the `QuizSplashActivity` class. Remember, costly operations, such as animations, should be stopped if the application is paused. The animation can resume when the application comes back into the foreground.

Let's start with the simplest case: applying the `slide_in_1` animation to your title `TextView` control, called `splashTitleTop`. All you need to do is retrieve an instance of your `TextView` control in the `onCreate()` method of the `QuizSplashActivity` class, load the animation resource into an `Animation` object, and call the `startAnimation()` method of the `TextView` control:

```

// hold objects for fast access
private TextView topTitle;

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

    topTitle = (TextView) findViewById(R.id.splashTitleTop);
}

```

```

        Animation slide1 = AnimationUtils
            .loadAnimation(this, R.anim.slide_in_1);

        topTitle.startAnimation(slide1);
    }

```

When an animation must be stopped—for example, in the `onPause()` callback method of the activity—you can simply call the `clearAnimation()` method. For instance, the following `onPause()` method implementation demonstrates this for the corner logos:

```

@Override
protected void onPause() {
    super.onPause();

    // stop animations
    if (topTitle != null) {
        topTitle.clearAnimation();
    }

    // ... stop other animations
}

```

Setting the Image Animations

The image animations are handled in a similar way, except that instead of loading them from the resources, the `createSplashAnim()` method is used. We can leverage the structure of the layout file to dynamically find each `ImageView` control. This gives us the flexibility to add more book covers later without changing the code.

Here's the code for finding the image objects, creating the animation object, applying the animation, and cleaning up the animations:

```

private ImageView[] splashImage;

@Override
public void onCreate(Bundle savedInstanceState) {
    // ...

    FrameLayout imagesHolder = (FrameLayout) findViewById(R.id.splashImages);
    int imgCount = imagesHolder.getChildCount();
    splashImage = new ImageView[imgCount];
    for (int i = 0; i < imgCount; i++) {
        splashImage[i] = (ImageView) imagesHolder.getChildAt(i);
        AnimationSet newAnim = createSplashAnim();
        splashImage[i].setAnimation(newAnim);
    }
}

```

```

@Override
protected void onPause() {
    super.onPause();
    // stop other animations ...
    if (splashImage != null) {
        for (ImageView img : splashImage) {
            img.clearAnimation();
        }
    }
}
}

```

At this point, after running the app in the emulator or on your Kindle Fire, you should see a screen similar to Figure 7.5. It will look different each time you run it. Try it!



Figure 7.5 The Have You Read That? Splash Screen After Animations Applied

Handling Animation Lifecycle Events

Now that your splash screen has some nice animations, all that's left is to handle the activity transition between `QuizSplashActivity` and `QuizMenuActivity` when the animations are complete. To do this, create a new `Intent` control to launch the `QuizMenuActivity`

class and pass it into the `startActivity()` method. Then call the `finish()` method of `QuizSplashActivity`, because you do not want to keep the `QuizSplashActivity` on the activity stack (that is, you do not want the Back button to return to the splash screen).

Of your animations, the `slide_in_2` animation finishes last. This animation is therefore the one you want to trigger your transition upon. You do so by creating an `AnimationListener` object, which has callbacks for the animation lifecycle events, such as `start`, `end`, and `repeat`. In this case, only the `onAnimationEnd()` method needs to be implemented; simply drop the code for starting the new `Activity` here. The following code shows how to create the `AnimationListener` and implement the `onAnimationEnd()` callback:

```
Animation slide2 = AnimationUtils.loadAnimation(this, R.anim.slide_in_2);
slide2.setAnimationListener(new AnimationListener() {
    @Override
    public void onAnimationEnd(Animation animation) {
        startActivity(new Intent(QuizSplashActivity.this,
            QuizMenuActivity.class));
        QuizSplashActivity.this.finish();
    }

    @Override
    public void onAnimationRepeat(Animation animation) {}

    @Override
    public void onAnimationStart(Animation animation) {}
});
```

Now, run the *Have You Read That?* application again, either on the emulator or on the handset. You now see some nice animation on the splash screen. The screen then smoothly transitions to the main menu, which is the next screen on your to-do list.

Summary

Congratulations! You've implemented the first screen of the *Have You Read That?* application. In this chapter, you designed a screen and then identified the appropriate layout and view components needed to implement your design. After you created the appropriate resources, you were able to configure the `splash.xml` layout file with various view controls, such as `TextView` and `ImageView`. Finally, you added some view animations to the screen and then handled the transition between `QuizSplashActivity` and `QuizMenuActivity`.

Exercises

1. Look at the documentation for `LayoutAnimationController` in the Android documentation (<http://goo.gl/Jjg48>). Create an animation resource and apply it to the `FrameLayout`. Remove the Java-defined animations. View the resulting animation.
2. Create a new animation resource. Apply it to any view you want, or maybe all of them. View the resulting animation.
3. [Challenging!] Design an alternative splash screen layout. Consider modifying the animation sequences to suit your alternative layout.

Implementing the Main Menu Screen

In this chapter, you learn about some of the different menu mechanisms available in Android. You begin by implementing the main menu screen of the Have You Read That? application, using layout controls, such as `RelativeLayout`. You also learn about a powerful control called a `GridView`, which is used to provide a variable-length scrolling list of items with individual click actions. Finally, you learn about other special types of menus available for use in your applications, such as the options menu.

Designing the Main Menu Screen

To design the main menu screen, begin by reviewing what its functions are and then roughly sketch what you want it to look like. If you review the screen requirements discussed in Chapter 6, “Designing an Application Framework,” you see that this screen provides essential navigation to the features of the rest of the application. Users can choose from four different options: play the game, review the help, configure the settings, or view the high scores. Figure 8.1 shows a rough design of the main menu screen.

There are a number of different ways you could implement the main menu screen. For example, you could create a button for each option, listen for clicks, and funnel the user to the appropriate screen. However, if the number of options grows, this method would not scale well. Therefore, a list of the options, in the form of a `GridView` control, is more appropriate. This way, if the list becomes longer than the screen, you have built-in scrolling capability. Each grid item will contain a handy icon for the screen it will transition to and text to indicate this.

Then you will wire up the `GridView` control to ensure that, when a user clicks a specific item, he or she is taken to the appropriate screen within the application. This allows users to access the rest of the screens you need to implement within the Have You Read That? application.

Recall that the full source code associated with this chapter can be downloaded from this book's websites.

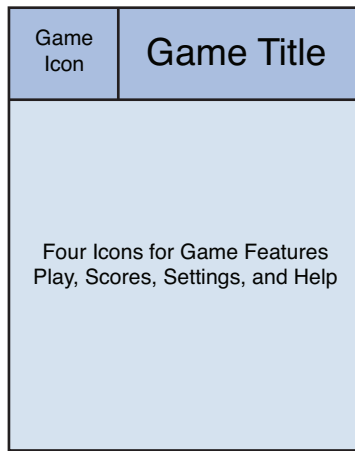


Figure 8.1 Rough Design for the Have You Read That? Main Menu Screen

Determining Main Menu Screen Layout Requirements

Now that you know how you want your main menu screen to look, you need to translate your rough design into the appropriate layout design. In this case, you need to update the `/res/layout/menu.xml` layout file that is used by `QuizMenuActivity`. In the case of the main menu layout, you want some sort of header followed by a `GridView` control.

Designing the Screen Header

The screen header has a small icon in the upper-left corner and then the title centered in the open space to the left of the icon. Therefore, you can describe the header inside a `RelativeLayout` as

- An `ImageView` control aligned to the top and left of the parent control
- A `TextView` control aligned to the top and right of the parent control and aligned to the bottom of the icon and to the icon's right

Designing the `GridView` Control

Next in your layout, include the `GridView` control. A `GridView` control is simply a container that holds a list of view objects. `GridView` is a type of control called an `AdapterView`. Controls derived from `AdapterView` are excellent for repeated items with identical layouts.

To effectively use the `GridView` control, an adapter must be created as well as the repeated layout.

Finishing Touches for the Main Menu Layout Design

Figure 8.2 shows the layout design of the main menu screen. The container `RelativeLayout` will receive the same background image as the splash screen. The `GridView` must also be placed below the header. The title is aligned to the bottom of the icon, which means the `GridView` can be placed below either. The `GridView` will take up the rest of the screen, scrolling if it needs to. (Scrolling won't be necessary with just four items on the Kindle Fire screen.)

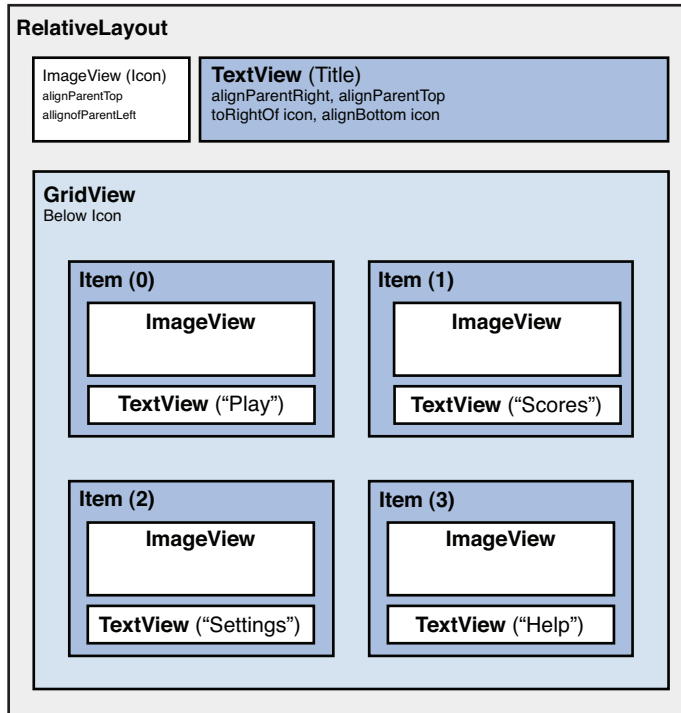


Figure 8.2 Layout Design for the Have You Read That? Main Menu Screen

Implementing the Main Menu Screen Layout

To implement the main menu screen, begin by adding new resources to the project. Then update the `menu.xml` layout resource to reflect the main menu screen design. Finally, add a `menu_item.xml` file to define the look of each grid item.

Adding New Project Resources

Now that you have your layout designed, you need to create the drawable, string, color, and dimension resources you will use in the layouts used by the main menu screen. For specific resource configurations, you can use the values provided in this book's source code as a guide or configure your own custom values.

Begin by adding four new graphic resources (in various resolutions) to the `/res/drawable` directory hierarchy: `game.png`, `help.png`, `scores.png`, and `settings.png`. Each of these is an icon for use with the grid. Also, add a `grid_background.xml` to the `/res/drawable` directory, with the following contents:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
    <solid
        android:color="#69000000" />
</shape>
```

This simply defines a translucent black drawable resource that will be used to replace the default orange highlight color on the grid when it is clicked. (Just setting it to a color doesn't work as of this writing; see <http://goo.gl/9CFHF>.)

Continue by adding and modifying several new strings in the `/res/values/strings.xml` resource file so that you have a string for each menu option, as well as one for the title `TextView` control. For example, the following string resources suffice:

```
<string name="menu_item_settings">Settings</string>
<string name="menu_item_play">Play</string>
<string name="menu_item_scores">Scores</string>
<string name="menu_item_help">Help</string>
```

Next, update the color resources in `/res/values/colors.xml` to include colors for the screen title `TextView` attributes and the `TextView` items displayed within the `GridView`. For example, we use the following color resources:

```
<color
    name="header_title_color">@color/splash_title_color</color>
<color
    name="menu_text_color">#494949</color>
```

Update the resources in `/res/values/dimens.xml` to include dimensions for the title text and the `GridView` item text. For example, the following dimension resource works well:

```
<dimen
    name="screen_title_size">48dp</dimen>
```

Next, create a `styles.xml` file. In it, we'll define a style for the text on each item of the grid. A style is merely a list of attributes and their values. The values can be literal or references

themselves. Here is the style we'll use, which shows mostly literal values, but also a color reference. In an app where values are likely to be frequently reused, reference values are recommended:

```
<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <style name="menuItemTextStyle" parent="android:Widget.TextView">
        <item name="android:shadowColor">#eeeeea</item>
        <item name="android:shadowDx">0</item>
        <item name="android:shadowDy">0</item>
        <item name="android:shadowRadius">5</item>
        <item name="android:textColor">@color/menu_text_color</item>
        <item name="android:textSize">40dp</item>
        <item name="android:textStyle">bold</item>
    </style>
</resources>
```

Finally, create a resource XML file named `grid_arrays.xml`. In it, add two array resources: One is an array of string references, and the other is an array of drawable references. These will be used to control the grid items and simplify its configuration. This is the entire file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="menu_items">
        <item>@string/menu_item_play</item>
        <item>@string/menu_item_scores</item>
        <item>@string/menu_item_settings</item>
        <item>@string/menu_item_help</item>
    </string-array>
    <array name="menu_item_images">
        <item>@drawable/game</item>
        <item>@drawable/scores</item>
        <item>@drawable/settings</item>
        <item>@drawable/help</item>
    </array>
</resources>
```

Save the resource files. Once it's saved, you can begin to use them in the layout resource files used by the main menu screen.

Updating the Main Menu Screen Layout Files

Perhaps you have noticed by now that the main menu screen relies on layout resource files—plural. The master layout file, `menu.xml`, defines the layout of the overall screen. You must separately create a new layout file used by the `GridView` control as a template for each item.

Updating the Master Layout

First, you need to update the master layout.

Note

The Eclipse layout resource editor does not always display complex controls or dynamic controls, like `ListView` or `GridView` controls, properly in design mode. Although it has a preview mode, editing in XML mode can be simpler. In this case, the layout designer does not reflect actual application look and feel.

Again, open the Eclipse layout resource editor and remove all existing controls from the `menu.xml` layout file. Then follow these steps to generate the main menu screen, based on your intended layout design:

1. Add a new `RelativeLayout` control and set its `background` attribute to `@drawable/background_paper`. All subsequent controls will be added inside this control.
2. Add an `ImageView` control called `headerIcon`. Set its `layout_alignParentTop` and `layout_alignParentLeft` attributes to `true`. Set the `src` attribute to `@drawable/ic_launcher`. Its `layout_width` and `layout_height` attributes should both be `wrap_content`.
3. Add a `TextView` control, called `title`, with `layout_width` and `layout_height` both set to `wrap_content`. Set its `layout_alignBottom` and `layout_toRightOf` attributes to `@id/headerIcon`. Set its `layout_alignParentRight` and `layout_alignParentTop` attributes to `true`.
4. Finally, add the `GridView`, simply called `gridView1`. Its `layout_width` and `layout_height` are also `wrap_content`. Set its `layout_below` attribute to `@id/headerIcon`. Set its `numColumns` attribute to 2 and `listSelector` to `@drawable/grid_background`.

At this point, save the `menu.xml` layout file. A full XML listing for this layout can be found in the sample code for Chapter 8, available for download from this book's websites.

Adding the GridView Template Layout

A `GridView` control has a variable number of items, where each item is displayed using a simple layout template. You now need to create this new layout resource for your project. For example, the `/res/layout/menu_item.xml` layout resource file can serve as a template for your `GridView` in the `menu.xml` layout resource. In this case, the `menu_item.xml` layout file will contain a `TextView` and `ImageView` controls to display the menu items (scores, help, and so on).

The `TextView` and `ImageView` controls require all the typical attributes assigned except for one: the text or image references. The `text` and `src` attributes will be supplied programmatically. (Optionally, these two attributes can be set to take advantage of the preview mode of the Graphical Layout tab.)

The `menu_item.xml` file looks like this:

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

    <ImageView
        android:id="@+id/menuItemImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:src="@drawable/game" />

    <TextView
        android:id="@+id/menuItemText"
        style="@style/menuItemTextStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_below="@id/menuItemImage"
        android:layout_centerHorizontal="true"
        android:text="@string/menu_item_play" />
</RelativeLayout>
```

At this point, save the `menu_item.xml` layout file. Optionally, at this point, right-click the `GridView` in the Graphical Layout tab. Choose `Preview Grid Content`, `Choose Layout`, and then pick `menu_item` in the Resource Chooser. The preview should now look something like Figure 8.3. As you can see, this is just a simulated view. However, it's helpful to get as much of the layout correct as possible before running on the device.

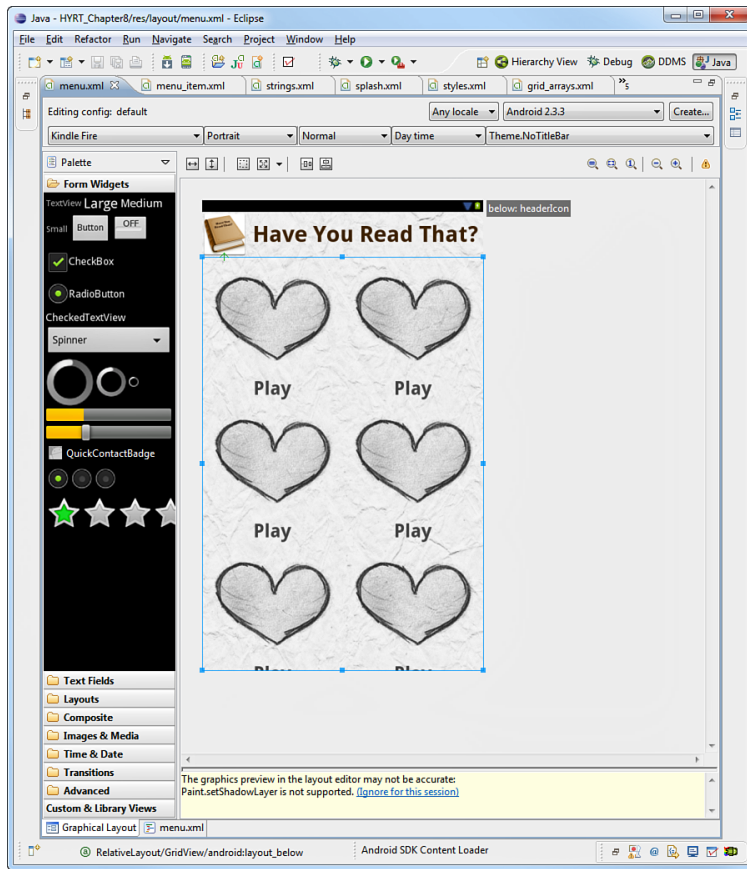


Figure 8.3 Layout Preview in the Graphical Layout View

Working with the GridView Control

Now it's time to switch your focus to the `QuizMenuActivity.java` file. Here, you need to wire up the `GridView` control. First, you need to fill the `GridView` control with content, and then you need to listen for user clicks on specific items in the `GridView` control and send the user to the appropriate screen in the application.

Filling a GridView Control

Your `GridView` control needs content. `GridView` controls can be populated from a variety of data sources, including arrays and databases, using data adapters. Because we're using a custom

item view with an image and a string, we'll create a custom adapter to wire up the arrays to the grid.

GridView setup occurs in the `onCreate()` method of the `QuizMenuActivity` class, just after the `setContentView()` method call. To populate your `GridView` control, you must first retrieve it by its unique identifier by using the `findViewById()` method, as follows:

```
GridView menuList = (GridView) findViewById(R.id.gridView1);
```

Next, the data adapter for the `GridView` needs to be set. The choice of adapter depends on the type of data being used. In this case, use the following data adapter, defined inside `QuizMenuActivity`:

```
class QuizMenuAdapter extends BaseAdapter {
    private String[] itemStrings;
    private TypedArray itemIcons;

    public QuizMenuAdapter(int stringArrayId, int imageArrayId) {

        itemStrings = getResources().getStringArray(stringArrayId);
        itemIcons = getResources().obtainTypedArray(imageArrayId);
    }

    @Override
    public int getCount() {
        return itemStrings.length;
    }

    @Override
    public Object getItem(int position) {
        // NA
        return null;
    }

    @Override
    public long getItemId(int position) {
        // NA
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View itemView = convertView;
        if (convertView == null) {
            itemView = getLayoutInflater()
                .inflate(R.layout.menu_item, null);
        }
    }
}
```

```

        ImageView image = (ImageView) itemView
            .findViewById(R.id.menuItemImage);
        TextView text = (TextView)
            itemView.findViewById(R.id.menuItemText);
        image.setImageDrawable(itemIcons.getDrawable(position));
        text.setText(itemStrings[position]);

        return itemView;
    }
}

```

This adapter loads the two arrays you already created when it is instantiated. The main behavior is in the `getView()` method, which is used to return the required view controls. If a view isn't being reused (when `convertView` is null), the `menu_item_layout` is loaded. The image and text are then set from the arrays and the modified—or new—view is returned.

Next, tell the `GridView` control to use this data adapter using the `setAdapter()` method:

```

QuizMenuAdapter adapter = new QuizMenuAdapter(R.array.menu_items,
    R.array.menu_item_images);
menuList.setAdapter(adapter);

```

At this point, save the `QuizMenuActivity.java` file and run the *Have You Read That?* application in the Android emulator or on your Kindle Fire. After the splash screen finishes, the main menu screen should look similar to the screen shown in Figure 8.4.

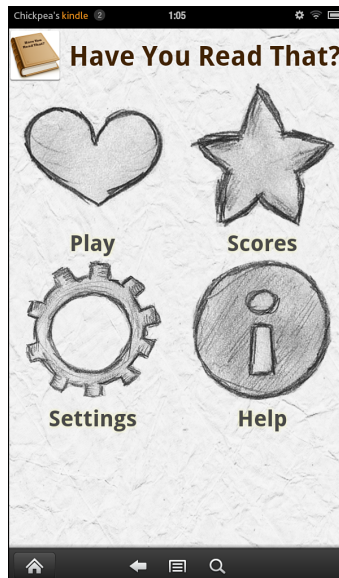


Figure 8.4 The *Have You Read That?* Splash Screen

As you see, the main menu screen is beginning to take shape. However, clicking the menu items doesn't yet have the desired response. Nothing happens!

Listening for GridView Events

You need to listen for and respond to specific events within the `GridView` control. Although there are a number of events to choose from, you are most interested in the event that occurs when a user clicks a specific menu item in the `GridView` control.

To listen for item clicks, use the `setOnItemClickListener()` method of the `GridView`. Specifically, implement the `onItemClick()` method of the `AdapterView.OnItemClickListener` class. Although there are many solutions, we chose to add a new method to the adapter class to help us:

```
public void onCreate(Bundle savedInstanceState) {
    // ...

    menuList.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View itemClicked,
                int position, long id) {
                QuizMenuAdapter adapter = (QuizMenuAdapter)
                    parent.getAdapter();
                Intent nextScreen = adapter.getItemIntent(position);
                if (nextScreen != null) {
                    startActivity(nextScreen);
                }
            }
        });
}

class QuizMenuAdapter extends BaseAdapter {

    //...

    public Intent getItemIntent(int position) {
        Intent intent = null;
        switch (position) {
            case 0:
                intent = new Intent(QuizMenuActivity.this,
                    QuizGameActivity.class);
                break;
            case 1:
                intent = new Intent(QuizMenuActivity.this,
                    QuizScoresActivity.class);
                break;
        }
    }
}
```

```

        case 2:
            intent = new Intent(QuizMenuActivity.this,
                               QuizSettingsActivity.class);
            break;
        case 3:
            intent = new Intent(QuizMenuActivity.this,
                               QuizHelpActivity.class);
            break;
    }

    return intent;
}
}

```

The `onItemClick()` method passes in all the information needed to determine which item was clicked. Here, we cast the parent to our adapter and call its `getIntent()` method. The `getIntent()` method knows that there are only four menu items and knows which order they are in. It uses this to return a newly created `Intent` object that is then used to switch screens.

Now rerun the application in the emulator. You can now use the main menu to transition between the screens in the *Have You Read That?* application.

Working with Other Menu Types

The Android platform has several other types of useful menu mechanisms, including the following:

- **Context menus**—A context menu pops up when a user performs a long-click on any view object. This type of menu is often used in conjunction with `ListView` controls filled with similar items, such as songs in a playlist. The user can then long-click a specific song to access a context menu with options such as Play, Delete, and Add to Playlist for that specific song.
- **Options menus**—An options menu pops up whenever a user clicks the Menu button on the device. Although this has changed substantially in Android 3 and later, the Kindle Fire has a traditional menu button—albeit, on the screen rather than a physical button—and so is treated the same. This type of menu is often used to help the user handle application settings and such.

Because we've been focusing on application screen navigation in this chapter, let's consider where these different menus are appropriate in the *Have You Read That?* application. This application design lends itself well to an options menu for the game screen, which would enable the user to pause while answering trivia questions to access the settings and help screens easily and then return to the game screen.

Adding an Options Menu to the Game Screen

To add an options menu to the game screen, you need to add a special type of resource called a menu resource. You can then update the `QuizGameActivity` class (which currently does nothing more than display a string of text saying it's the game screen) to enable an options menu and handle menu selections.

Adding Menu Resources

For your options menu, create a menu definition resource in XML and save it to the `/res/menu` resource directory as `gameoptions.xml`.

A menu resource is a special type of resource that contains a `<menu>` tag followed by a number of `<item>` child elements. Each `<item>` element represents a menu option and has a number of attributes. The following are some commonly used attributes:

- `id`—Allows you to easily identify the specific menu item
- `title`—The string shown for the options menu item
- `icon`—A drawable resource representing the icon for the menu item

Your options menu will contain only two options: Settings and Help. Therefore, your `gameoptions.xml` menu resource is fairly straightforward:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/settings_options_menu_item"
        android:icon="@android:drawable/ic_menu_preferences"
        android:title="@string/menu_item_settings">
    </item>
    <item
        android:id="@+id/help_options_menu_item"
        android:icon="@android:drawable/ic_menu_help"
        android:title="@string/menu_item_help">
    </item>
</menu>
```

Set the `title` attribute of each menu option by using the same `String` resources you used on the main menu screen. Note that instead of adding new drawable resources for the options menu icons, you use built-in drawable resources from the Android SDK to have a common look and feel across applications.

Adding an Options Menu to an Activity

For an options menu to show when the user presses the Menu button on the game screen, you must provide an implementation of the `onCreateOptionsMenu()` method in the `QuizGameActivity` class. Specifically, you need to inflate (load) the menu layout resource into the options menu and set the appropriate `Intent` information for each menu item. Here is a sample implementation of the `onCreateOptionsMenu()` method for you to add to `QuizGameActivity`:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    getMenuInflater().inflate(R.menu.gameoptions, menu);
    menu.findItem(R.id.help_options_menu_item).setIntent(
        new Intent(this, QuizHelpActivity.class));
    menu.findItem(R.id.settings_options_menu_item).setIntent(
        new Intent(this, QuizSettingsActivity.class));
    return true;
}
```

Handling Options Menu Selections

To listen for when the user launches the options menu and selects a menu option, implement the `onOptionsItemSelected()` method of the activity. For example, start the appropriate activity by extracting the intent from the menu item selected, as follows:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);
    startActivity(item getIntent());
    return true;
}
```

There you have it: You have created an options menu on the game screen. This method for handling `onOptionsItemSelected()` works as designed. It's not technically required if the only thing your menu will do is launch the `Intent` set via the `setIntent()` method. However, to add any other functionality to each `MenuItem` requires the implementation of this method.

Save your changes and run the application once more. Navigate to the game screen, press the Menu button, and see that you can now use a fully functional options menu (see Figure 8.5).

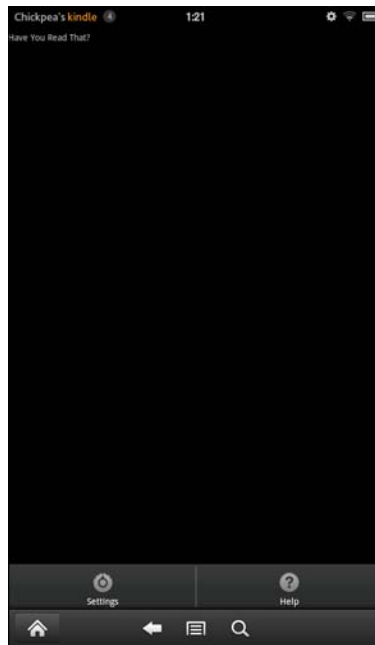


Figure 8.5 The Have You Read That? Game Screen with an Options Menu

Summary

You've made excellent progress. The main menu screen of the Have You Read That? application is now fully functional. You learned important skills for developing Android applications, including how to use layouts such as `RelativeLayout`, as well as how to use the powerful `GridView` control. You also learned about the other types of navigation mechanisms available in Android and implemented via an options menu on the game screen.

Exercises

1. Add a third option to the game screen's options menu to allow the user to access the scores screen.
2. Modify the outer `RelativeLayout` control of `menu.xml` to include an animation that brings in some of the book-cover images from the splash screen and places them scattered about along the bottom of the screen.

This page intentionally left blank

Developing the Help and Scores Screens

In this chapter, you implement two more screens of the Have You Read That? application: the help and scores screens. You begin by implementing the help screen using a `TextView` control with text supplied from a text file, which allows you to explore some of the file support classes of the Android SDK. Next, you design and implement the scores screen. With its more complicated requirements, the scores screen is ideal for trying out the tab set control called `TabHost`. Finally, you test the scores screen by parsing XML score data.

Designing the Help Screen

The help screen requirements are straightforward: This screen must display a large quantity of text and should have scrolling capabilities. Figure 9.1 shows a rough design of the help screen.

For consistency and familiarity, application screens share some common features. Therefore, the help screen mimics some of the menu screen features, such as a header. To translate your rough design into the appropriate layout design, update the `/res/layout/help.xml` layout file and the `QuizHelpActivity` class.

Use the same title header you used in the menu screen (using a `RelativeLayout`), followed by a `TextView` control within a `ScrollView` control. Figure 9.2 shows the layout design for the help screen.

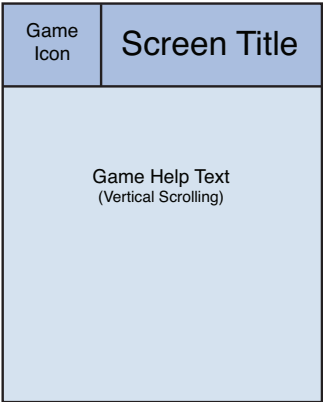


Figure 9.1 Rough Design for the Have You Read That? Help Screen

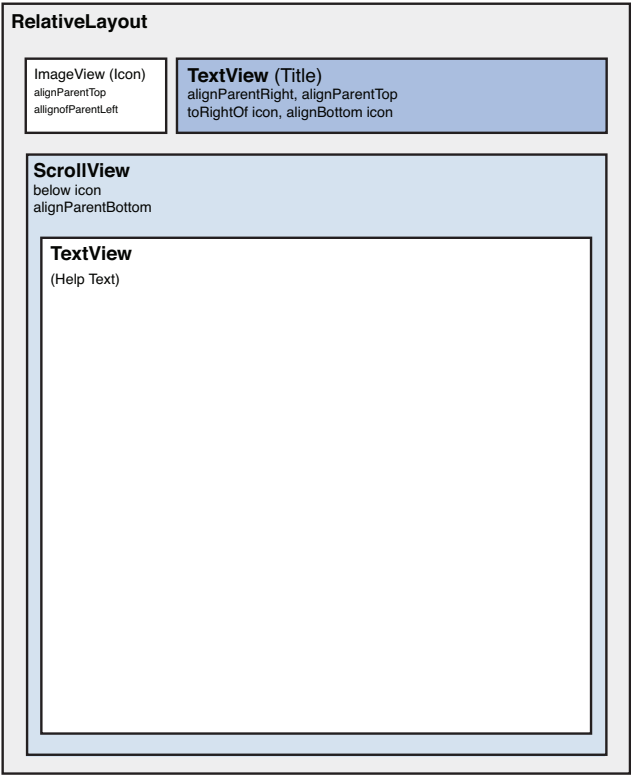


Figure 9.2 Layout Design for the Have You Read That? Help Screen

Implementing the Help Screen Layout

To implement the help screen, begin by adding new resources to the project. Then, update the `help.xml` layout resource to reflect the help screen design.

Adding New Project Resources

As with the other screens in the Have You Read That? application, you need to add numerous string, color, and dimension resources to your project to support the help screen. Specifically for this implementation, you may want to add several dimension resources in `/res/values-large/dimens.xml` for the help attributes (we talk more about the different values' folder names in Chapter 15, "Managing Alternative and Localized Resources"):

```
<dimen
    name="help_text_padding">20dp</dimen>
<dimen
    name="help_text_size">24sp</dimen>
<dimen
    name="help_text_drawable_padding">5dp</dimen>
```

Save the dimension resource file. The new dimensions can now be used in the layout resource files used by the help screen.

In addition to these support resources you use within the layout for the help screen, add a new type of resource: a raw file resource file. Create a text file called `/res/raw/quizhelp.txt` that includes a number of paragraphs of help text to display in the main `TextView` control of the help screen.

Updating the Help Screen Layout

The `help.xml` layout file dictates the user interface of the help screen. Follow these steps to generate the help screen layout, based on the screen design:

1. Open the Eclipse layout resource editor and remove all existing controls from the layout resource file.
2. Add a `RelativeLayout` control and set its `background` attribute to `@drawable/background_paper`. Set its `layout_width` and `layout_height` attributes both to `match_parent` to fill the screen. All subsequent controls will be added inside the `RelativeLayout` control.
3. Add the same header you created in the `menu.xml` layout. It is simply an `ImageView` control and a `TextView` control. Set the `TextView` control's `text` attribute to the string resource called `@string/help` to reflect the appropriate screen title.
4. Add a new `TextView` control, setting the `layout_width` attribute to `match_parent` and the `layout_height` attribute to `wrap_content`. Set the `textSize` and padding values to use the dimension resource values previously defined.

Note

You can make text in a `TextView` control bold or italic by using the `textStyle` attribute. In the source code example provided, we make the help text italic using this handy attribute.

5. The `TextView` control may contain more text than can be displayed on the screen. The `ScrollView` control solves this problem by allowing its children to grow virtually beyond its own bounds by providing a scrollable area and a scrollbar to indicate the scrolling. To give a `TextView` control a vertical scrollbar and vertical-scrolling capability, wrap it in a `ScrollView` control and set the `scrollbars` attribute to `vertical`. Then, set `layout_below` to `headerIcon` and `layout_alignParentBottom` to `true`.

TextView XML Attributes

The Android SDK documentation for the XML attributes for `TextView` controls can be found at <http://goo.gl/a1N2T>. You may also have to look at the attributes for `View` controls for some of the inherited attributes, like the scrollbar attributes.

At this point, save the `help.xml` layout file.

Working with Files

Each Android application has its own private directory on the Android file system for storing application files. In addition to all the familiar `File` (`java.io.File`) and `Stream` (`java.io.Stream`) classes available, you can access private application files and directories by using the following `Context` class methods: `fileList()`, `getFilesDir()`, `getDir()`, `openFileInput()`, `openFileOutput()`, `deleteFile()`, and `getFilePath()`. These features can be helpful if your application needs to generate files or download them from the Internet.

Now that the `help.xml` layout file is complete, the `QuizHelpActivity` class must be updated to read the `quizhelp.txt` file and place the resulting text into the `TextView` control called `helpText`.

Adding Raw Resource Files

Raw resource files, such as the `quizhelp.txt` text file, are added to a project by simply including them in the `/raw` resources project directory. This can be done by creating them as a new file, dragging them in from a file-management tool, or any other way you're accustomed to adding files to Android projects in Eclipse.

For the purposes of this exercise, we created a text file that contained some basic help text and copyright information, as well as a website, email address, street address, and phone number. This text file is included in the source code for this chapter for you to use.

Accessing Raw File Resources

The Android platform includes many of the typical Java file I/O classes, including stream operations. To read string data from a file, use the `openRawResource()` method of the `Resources` class from within your activity, as in the following example:

```
InputStream helpFileStream = getResources().openRawResource(R.raw.quizhelp);
```

Now that you have an `InputStream` object, you can read the file, line-by-line or byte-by-byte, and create a string. There are a number of ways to do this in Java. Here's a simple Java method that reads an `InputStream` and returns a `String` with its contents:

```
public String inputStreamToString(InputStream is) throws IOException {
    StringBuffer sBuffer = new StringBuffer();
    DataInputStream dataIO = new DataInputStream(is);
    String strLine = null;
    while ((strLine = dataIO.readLine()) != null) {
        sBuffer.append(strLine + "\n");
    }

    dataIO.close();
    is.close();

    return sBuffer.toString();
}
```

This helper method should be used within a `try/catch` block. (See this chapter's sample code if you require further explanation.) Use the `inputStreamToString()` method with the `InputStream` of the help file to retrieve the help text. Then, retrieve the `TextView` control using the `findViewById()` method and set the help text to it using the `TextView` control's `setText()` method, as follows:

```
TextView helpText = (TextView) findViewById(R.id.helpText);
String strFile = inputStreamToString(helpFileStream);
helpText.setText(strFile);
```

At this point, save the `QuizHelpActivity.java` file and run the Have You Read That? application in the Android emulator. After the splash screen finishes, choose the help screen option from the main menu. The help screen should now look like Figure 9.3.

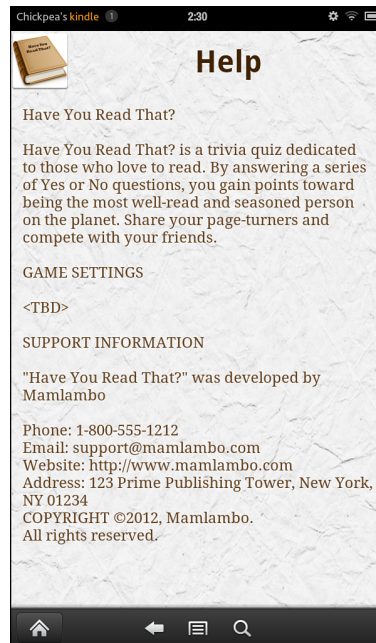


Figure 9.3 The Have You Read That? Help Screen

Designing the Scores Screen

Now that you created the help screen, it's time to turn your attention to another screen: the scores screen. The requirements for this screen include showing several different scores to the user. There are two types of scores: the all-time-high scores and the user's friends' scores. The same screen handles both categories of scores. Each user shown includes their name, score, and overall ranking.

There are a number of ways you could implement the scores screen. For example, you could use string formatting with a `TextView` control or `ListView` control to display the score information. However, you are working with a small screen, and you don't want to overwhelm the user with too much information. Because you have two different sets of data to display, two tabs would be ideal for this screen. Figure 9.4 shows a rough design of the scores screen.

Determining Scores Screen Layout Requirements

Now that you have the rough design of the scores screen, translate the design to use the appropriate layout controls. To do this, you update the `/res/layout/scores.xml` layout file that is used by the `QuizScoresActivity` class. Again, take advantage of the `RelativeLayout`

control to add a familiar title bar to the top of the scores screen. This header will be followed by a `TabHost` control with two tabs: one tab for all user scores and one for friends' scores. Each tab contains a `TableLayout` control to display scores in neat rows and columns. Although we could have used a `ListView`, this is as good a place as any to teach you about the `TableLayout` control—you already learned about the `ListView` control in Chapter 8, “Implementing the Main Menu Screen.”

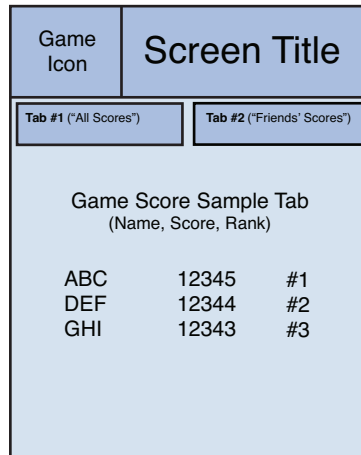


Figure 9.4 Rough Design for the Have You Read That? Scores Screen

Adding the `TabHost` Control

To add tabbing support to the scores screen, you must include a `TabHost` control, which is a container view with child tabs, each of which may contain layout content. The `TabHost` control is a somewhat complex object, and you may want to review the Android SDK documentation regarding this class if you run into problems or require clarification about how to properly configure it, above and beyond the steps discussed here. To configure tab controls within an XML layout resource file, follow these guidelines:

- Include a `TabHost` control.
- Ensure that there is a `LinearLayout` within the `TabHost` control.
- Ensure that there is a specially named `TabWidget` control and `FrameLayout` control within the `LinearLayout` control.
- Define the contents of each tab in a `FrameLayout` control.

Figure 9.5 shows the layout design for the scores screen.

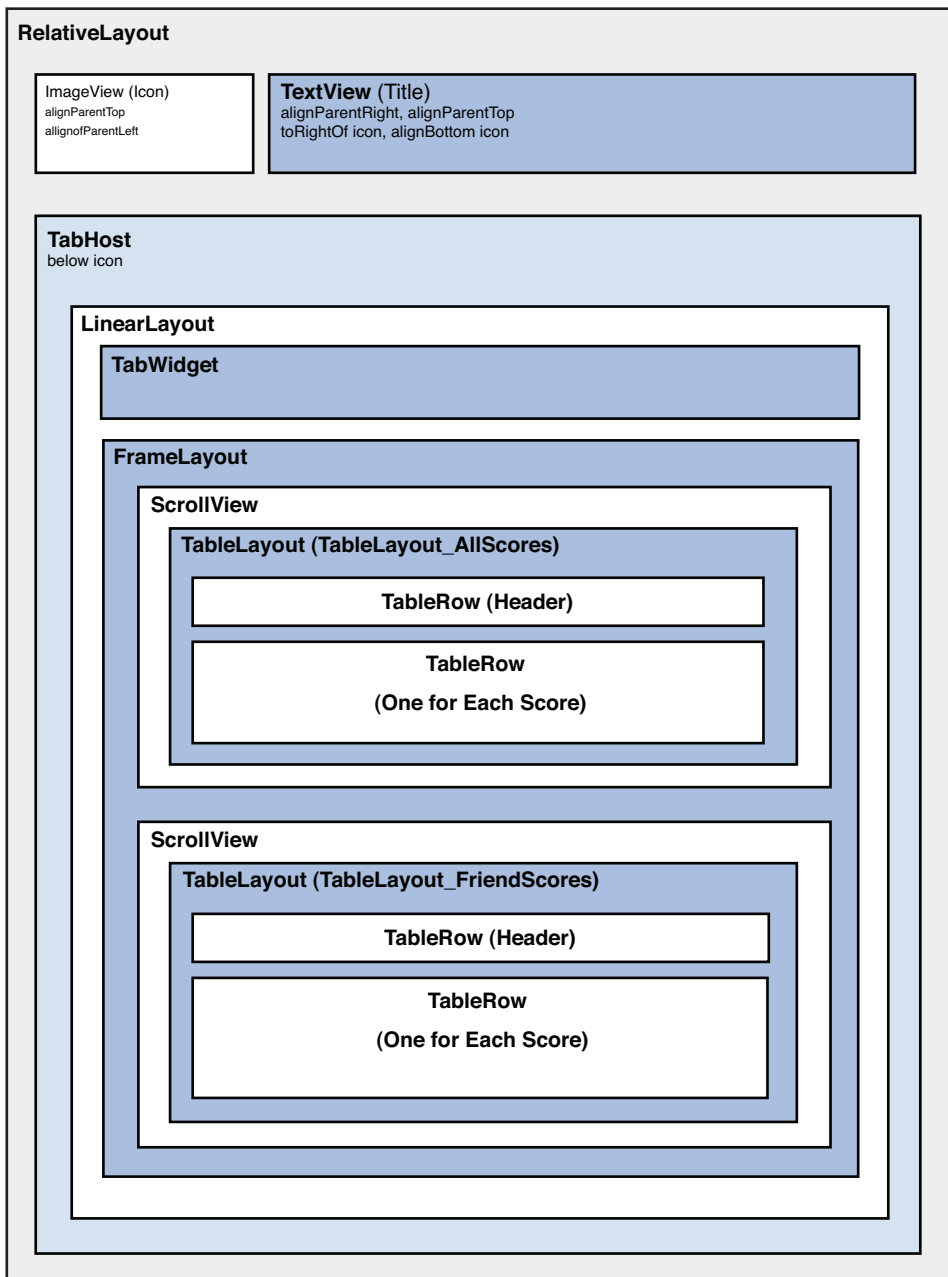


Figure 9.5 Layout Design for the Have You Read That? Scores Screen

Implementing the Scores Screen Layout

To implement the scores screen, begin by adding new resources to the project. Then, update the `scores.xml` layout resource to reflect the scores screen design. Let's walk through how to do each of these tasks now.

Adding New Project Resources

As with the other screens in the Have You Read That? application, you need to add several new string, color, and dimension resources to your project to support the scores screen. Start by adding string resources to `/res/values/strings.xml` for the score column names, status string, and when no scores exist. We used the following strings:

```
<string
    name="all_scores">"All Scores"</string>
<string
    name="friends_scores">"Scores of Friends"</string>
<string
    name="no_scores">"No scores to show."</string>
<string
    name="username">"Username"</string>
<string
    name="rank">"Ranking"</string>
<string
    name="score">"Score"</string>
<string
    name="wait_msg">"Retrieving Scores..."</string>
<string
    name="wait_title">"Loading..."</string>
```

Save the string resource file. Now these strings are available for use in the scores screen layout resource file.

The scores for the Have You Read That? application will eventually be retrieved from a remote network server, but we've got to take baby steps for now so instead, for this chapter, build the screen using some mock score data. Android supports the resource file type needed. XML resource files can contain this mock score data, so you can mimic the structure you will use when the real network scores are available.

Therefore, add two files to the `/res/xml/` resource directory—`allscores.xml` and `friendscores.xml`—that represent the mock score data. These files have the following XML structure:

```
<?xml version="1.0" encoding="utf-8"?>
  <!-- This is a mock score XML chunk -->
<scores>
  <score
    username="ESC"
    score="12346"
```

```

        rank="1" />
    <score
        username="LED"
        score="12345"
        rank="2" />
    <score
        username="SAC"
        score="12344"
        rank="3" />
</scores>

```

The score data uses a very simple schema. A single `<scores>` element has a number of child `<score>` elements. Each `<score>` element has three attributes: `username`, `score`, and `rank`. For this example, assume that the score data will be sorted and limited to the top 20 or so scores. A server for this data will later enforce these restrictions.

Updating the Scores Screen Layout

The scores screen user interface is defined in the `scores.xml` layout file.

Previewing `TabHost` Controls

This screen design uses `TabHost` controls. The Eclipse layout resource editor does not display `TabHost` controls properly in design mode—it throws a `NullPointerException`. To design this kind of layout, you should stick to the XML layout mode. You must use the Android emulator or an Android device to view the tabs.

To update this layout to your intended layout design, follow these steps:

1. Remove all the old controls and start fresh.
2. Add a new `RelativeLayout` control, setting its `android:background` attribute to `@drawable/background_paper`. Set its `layout_width` and `layout_height` attributes to `match_parent` to fill the screen. All subsequent controls are added inside this `RelativeLayout` control.
3. Add the same header you created in other layouts. Recall that it contains an `ImageView` control and a `TextView` control. Set the `TextView` control's `text` attribute to the string resource `@string/scores` to reflect the appropriate screen title.
4. Add a `TabHost` control with an `id` attribute of `@+id/scoresTabHost`. Set its `layout_width` and `layout_height` attributes to `match_parent`.
5. Inside the `TabHost` control, add another `LinearLayout` control, with its `orientation` attribute set to `vertical`. Set its `layout_width` and `layout_height` attributes to `match_parent`.

6. Inside the inner `LinearLayout` control, add a `TabWidget` control. Set the control's `id` attribute to `@android:id/tabs`, its `layout_width` to `match_parent`, and its `layout_height` to `wrap_content`. The identifier is mandatory; it must be exactly as shown.
7. Within the inner `LinearLayout` control at the same level as the `TabWidget` control, add a `FrameLayout` control. Set the `FrameLayout` control's `id` attribute to `@android:id/tabcontent` and its `layout_width` and `layout_height` attributes to `match_parent`. When creating a tabbed view in this way, you must name the `FrameLayout` control `@android:id/tabcontent`; otherwise, exceptions will be thrown at runtime. This identifier is expected by the `TabHost` control and references a special Android package resource. It is not the same as using `@id/tabcontent`. That would create a new identifier for a layout object in your own application package. Thus, the identifier is mandatory and must appear exactly as shown.
8. Define the content of your tabs. Within the `FrameLayout` control, add two `TableLayout` controls, one for each tab. The scores are displayed in neat rows and columns using these `TableLayout` controls. Name the first `TableLayout` control `allScoresTable` and the second `friendScoresTable`. Set each `TableLayout` control's `layout_width` to `match_parent` and `layout_height` attributes to `wrap_content`. Set the `stretchColumns` attribute to `*` to allow columns to resize based on the content.
9. The list of scores may grow longer than the available vertical space on the screen. To give a `TableLayout` control a vertical scrollbar and vertical-scrolling capability, wrap it in a `ScrollView` control (inside the `FrameLayout`, encompassing a single `TableLayout`) and set the `scrollbars` attribute to `vertical`. You also need to set its `layout_width` and `layout_height` attributes.

The `TabHost` section of the scores screen layout file (with optional scrolling `TableLayout` tabs) should now look something like this:

```
<TabHost
    android:id="@+id/scoresTabHost"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/headerIcon" >
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TabWidget
        android:id="@android:id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <FrameLayout
```

```

        android:id="@android:id/tabcontent"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <ScrollView
            android:id="@+id/allScoresContent"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="vertical" >
            <TableLayout
                android:id="@+id/allScoresTable"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:stretchColumns="*" >
            </TableLayout>
        </ScrollView>
        <ScrollView
            android:id="@+id/friendScoresContent"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="vertical" >
            <TableLayout
                android:id="@+id/friendScoresTable"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:stretchColumns="*" >
            </TableLayout>
        </ScrollView>
    </FrameLayout>
</LinearLayout>
</TabHost>

```

Save the `scores.xml` layout file.

Building a Screen with Tabs

It's time to switch your focus to the `QuizScoresActivity.java` file and wire up the controls needed by the `TabHost` control. First, initialize the `TabHost` control and add the two tabs, making the default tab the All Scores tab. Finally, parse the mock XML score data and populate the `TableLayout` control for each tab. Let's now discuss how to do these tasks.

Configuring the `TabHost` Control

The `TabHost` control must be initialized before it will function properly. Therefore, start by retrieving the control by using the `findViewById()` method. Next, call the `TabHost` control's

`setup()` method to initialize the `TabHost` and “glue” the specially named `TabWidget` and `FrameLayout` controls together to form a tab set, as follows:

```
TabHost host = (TabHost) findViewById(R.id.scoresTabHost);
host.setup();
```

Adding Tabs to the `TabHost` Control

Now that the `TabHost` control is initialized, configure each tab and add the configured tabs to the `TabHost` using the `addTab()` method. The `addTab()` method takes a `TabSpec` parameter to describe the tab contents. To create the All Scores tab, add the following code right after the `setup()` method call:

```
TabSpec allScoresTab = host.newTabSpec("allTab");
allScoresTab.setIndicator(getResources().getString(R.string.all_scores),
    getResources().getDrawable(android.R.drawable.star_on));
allScoresTab.setContent(R.id.allScoresContent);
host.addTab(allScoresTab);
```

The `TabSpec` control called `allScoresTab` has the tag spec reference of `allTab`. The actual tab label contains both a `TextView` control label and a drawable icon (in this case, a star from the built-in Android resources). Finally, the contents of the tab are set to `allScoresContent` using a call to the `setContent()` method, which contains the `TableLayout` control called `allScoresTable`, defined in the `scores.xml` layout resource.

Implement the tab for friends’ scores using this same mechanism. (The sample code for this chapter uses `friendsTab` as the `TabSpec` name.) Change the content around to use the appropriate label for the tab indicator and the appropriate content with the `setContent()` method.

Setting the Default Tab

At this point, you need to identify which tab to show by default. To do this, call the `setCurrentTabByTag()` method and pass in the tag name of the tab you want to display by default. For example, to display the `allScoresTab` first, use the following method call, placed after the code for adding the tabs to the `TabHost`:

```
host.setCurrentTabByTag("allTab");
```

Save the `QuizScoresActivity.java` file and try to run the application in the Android emulator. Navigate to the scores screen. You should see the two tabs and blank space beneath. Let’s now fill that in with the scores.

Working with XML

The Android platform has a number of mechanisms for working with XML data, including support for the following:

- SAX (Simple API for XML)
- XML Pull Parser
- Limited DOM Level 2 core support

The XML technology you use depends on your specific project. For this example, you simply want to read through a simple XML file and extract the mock score data.

Retrieving XML Resources

First, write code to access the mock XML data you saved in the project resources. The Android SDK includes an easy method to retrieve XML resources into an object type that is used to parse the XML files: the `XMLResourceParser` object. Initialize two instances of this object, one for each score file, using the following code:

```
XmlResourceParser mockAllScores =
    getResources().getXml(R.xml.allscores);
XmlResourceParser mockFriendScores =
    getResources().getXml(R.xml.friendscores);
```

Now you've got an `XMLResourceParser` object that is used to parse the XML.

Parsing XML Files with `XmlResourceParser`

The mock score files have a simple schema with only two tags: `<scores>` and `<score>`. To parse the file, you want to find each `<score>` tag and extract its `username`, `rank`, and `score` attributes. Because you can assume a small amount of data (we guarantee it here), implement your parsing routine by using a simple `while()` loop to iterate through the events by using the `next()` method, as follows:

```
int eventType = -1;
boolean bFoundScores = false;
// Find Score records from XML
while (eventType != XmlResourceParser.END_DOCUMENT) {
    if (eventType == XmlResourceParser.START_TAG) {
        // Get the name of the tag (eg scores or score)
        String strName = scores.getName();
```

```

        if (strName.equals("score")) {
            bFoundScores = true;
            String scoreValue = scores.getAttributeValue(null, "score");
            String scoreRank = scores.getAttributeValue(null, "rank");
            String scoreUserName =
                scores.getAttributeValue(null, "username");
            insertScoreRow(scoreTable, scoreValue, scoreRank,
                scoreUserName);
        }
    }
    eventType = scores.next();
}

```

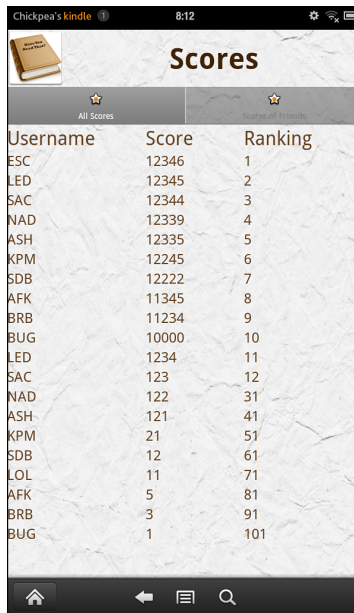
Within the loop, watch for the `START_TAG` event. When the tag name matches the `<score>` tag, a piece of score data is ready. Extract the score data by using the `getAttributeValue()` method. For each score, add a new `TableRow` control to the appropriate `TableLayout` control (in the appropriate tab); in this case, we implemented a helper method called `insertScoreRow()`. This method simply creates a new `TableRow` control with three new `TextView` controls (username, score, ranking) and adds the row to the `TableLayout` using the `addView()` method. (For the complete implementation of this helper method, see the source code that accompanies this chapter.)

Now, we said that this method would work for small amounts of data, and it does. But, when you have time-intensive processing, always perform the hard work asynchronously to the main thread. We discuss methods of doing this later in the book, but it's worth noting now that parsing is just such an operation, no matter how long it takes. For this chapter, we keep it simple.

Applying Finishing Touches to the Scores Screen

After you have written the code to parse the two mock XML files and populate the two `TableLayout` controls in the `TabHost` control, you need only make a few minor additions to `QuizScoresActivity`. Add a header `TableRow` to each `TableLayout` control, with nicely styled column headers using the string resources that you created earlier in this chapter. Then, implement special handling for the case where no score data is available. These tasks are a little different from populating the rows with scores; you're simply getting the text data from a different source.

When you're done applying these finishing touches, save the class and run the application in the emulator or on the device. Navigate to the scores screen. Both tabs are now populated with data and look similar to Figure 9.6.



Username	Score	Ranking
ESC	12346	1
LED	12345	2
SAC	12344	3
NAD	12339	4
ASH	12335	5
KPM	12245	6
SDB	12222	7
AFK	11345	8
BRB	11234	9
BUG	10000	10
LED	1234	11
SAC	123	12
NAD	122	31
ASH	121	41
KPM	21	51
SDB	12	61
LOL	11	71
AFK	5	81
BRB	3	91
BUG	1	101

Figure 9.6 The Have You Read That? Scores Screen

Summary

You've made excellent progress on building the Have You Read That? application in this chapter, including the implementation of two new screens. As you implemented the help screen, you learned how to display large amounts of data by adding a `ScrollView` wrapper around a `TextView` control. You learned how to access a file resource and change layout characteristics programmatically. By implementing the scores screen, you learned about the `TabHost` control, the `TableLayout` control, and even how to parse XML to display some mock score data to the screen.

Exercises

1. Change the indicator icon used by the All Scores tab to another drawable resource, either another built-in resource (such as `star_big_on`) or a drawable resource you supply to the project.
2. Experiment with the scrollbars implemented on both the help and scores screens. The `ScrollView` control has numerous scrollbar-related attributes that can be configured in different ways, colors, and styles. Try out some of them within your application. Which scrollbar style do you prefer?
3. Modify the help text typeface, color, and style using the appropriate `TextView` attributes.

Collecting User Input

In this chapter, you begin implementation of the settings screen of the Have You Read That? application. The settings screen displays a form for entering application configuration information, including the user's login and profile settings. Different settings necessitate the use of different input controls, including `EditText`, `Spinner`, and `Button` controls, among others. Finally, you need to ensure that each setting is saved and stored in a persistent manner as part of the application's preferences.

Designing the Settings Screen

The settings screen must allow the user to configure any number of game settings. Game settings may be text-input fields, dropdown lists, or other more complex controls. You will eventually need to also handle the social gaming settings, but we will deal with this requirement in a later chapter. For now, begin by implementing a simple settings screen with five basic game settings:

- **Nickname**—The name to be displayed on score listings. This text field should be no more than 20 characters long—an arbitrary but reasonable length for the purposes of this application.
- **Email**—The unique identifier for each user. This is a text field.
- **Password**—A mechanism to handle user verification. This is a password text field. When setting the password, the user should input the password twice for verification. The password text may be stored as plaintext.
- **Date of Birth**—To verify minimum age, when necessary. This is a date field, but it's often displayed in a friendly way that users can understand and easily configure.
- **Gender**—A piece of demographic information, which could be used for special score listings or to target ads to the user. This can be set to three different settings: Male (1), Female (2), or Prefer Not to Say (0).

Figure 10.1 shows a rough design for the settings screen.

Game Icon	Screen Title
NICKNAME: (20 Characters Max)	
Email: (Will Be Used as Unique Account ID)	
PASSWORD: (Password Requires Entering Twice to Verify)	
BIRTH DATE: (DOB Requires Entering Month, Day, Year)	
GENDER: (Male, Female, or Prefer Not to Say)	

Figure 10.1 Rough Design for the Have You Read That? Settings Screen

The application settings screen will contain many different controls, so you need to be especially careful with screen real estate. Begin with the customary header bar that contains the title of the screen.

Below the title, add a `ScrollView` control to contain all of the settings. This way, when the settings controls no longer fit on a single screen, users can easily scroll up and down to find the setting they require. A `ScrollView` control can have only a single child control, so place a vertical `LinearLayout` control within it to align the settings within.

Each setting requires two “rows” in the `LinearLayout` control: a `TextView` row that displays the setting name label and a row for the input control to capture its value. For example, the Nickname setting would require a row with a `TextView` control to display the label string (“Nickname:”) and a second row for an `EditText` control to allow the user to input a string of text.

Now determine which input control is most appropriate for each setting:

- The Nickname and Email fields are simply different types of single-line text input, so they can be `EditText` controls.
- The Password setting requires two `EditText` controls to request the password and confirm it. Use a `Dialog` object for these two input controls. This way, the entries aren’t

shown on the settings screen and don't take up extra room on the screen. The main settings screen can just display whether or not the password has been set in a simple `TextView` control and a `Button` control to launch the password dialog.

- The Birth Date setting requires a `DatePicker` input control. Because the `DatePicker` control is actually three separate controls—a month picker, a day picker, and a year picker—it takes up a lot of space on the screen. Therefore, instead of including it directly on the settings screen, you can add a `Button` control to launch a `DatePickerDialog` control in a dialog. The user then selects the appropriate date from the picker and closes the dialog. The resulting date is then displayed (but not editable) on the settings screen using a simple `TextView` control.
- The Gender setting is simply a choice between three values, so a `Spinner` (dropdown) control is most appropriate.

Figure 10.2 shows the layout design of the basic settings screen.

PreferenceScreen and PreferenceActivity

If you've looked through the Android SDK documentation, you might be wondering why these settings aren't implemented with a `PreferenceActivity` class and `PreferenceScreen` XML resource file. While there are several reasons, we also wanted a reasonable way of showing many of the user input controls. If you don't want to store settings in the `SharedPreferences`, you wouldn't want to use a `PreferenceScreen`. If you want a custom look, rather than the system look for preferences, you also wouldn't want to use a `PreferenceScreen`. Much of the time, using a `PreferenceScreen` works great, and users are familiar with it. If you use fragments in your application, the `PreferenceFragment` class can be used.

Implementing the Settings Screen Layout

To implement the settings screen, begin by adding new resources to the project. Then, update the `settings.xml` layout resource to reflect the settings screen design. In this chapter, you focus on the controls specific to the settings screen, but you won't implement the `Dialog` controls for the password and date picker until the next chapter.

Adding New Project Resources

Screens with form fields seem to rely on more resources than most other screen types. You need to add a number of new resources to support the settings screen. In addition to the string and color resources, you need to add a new type of resource: a string array.

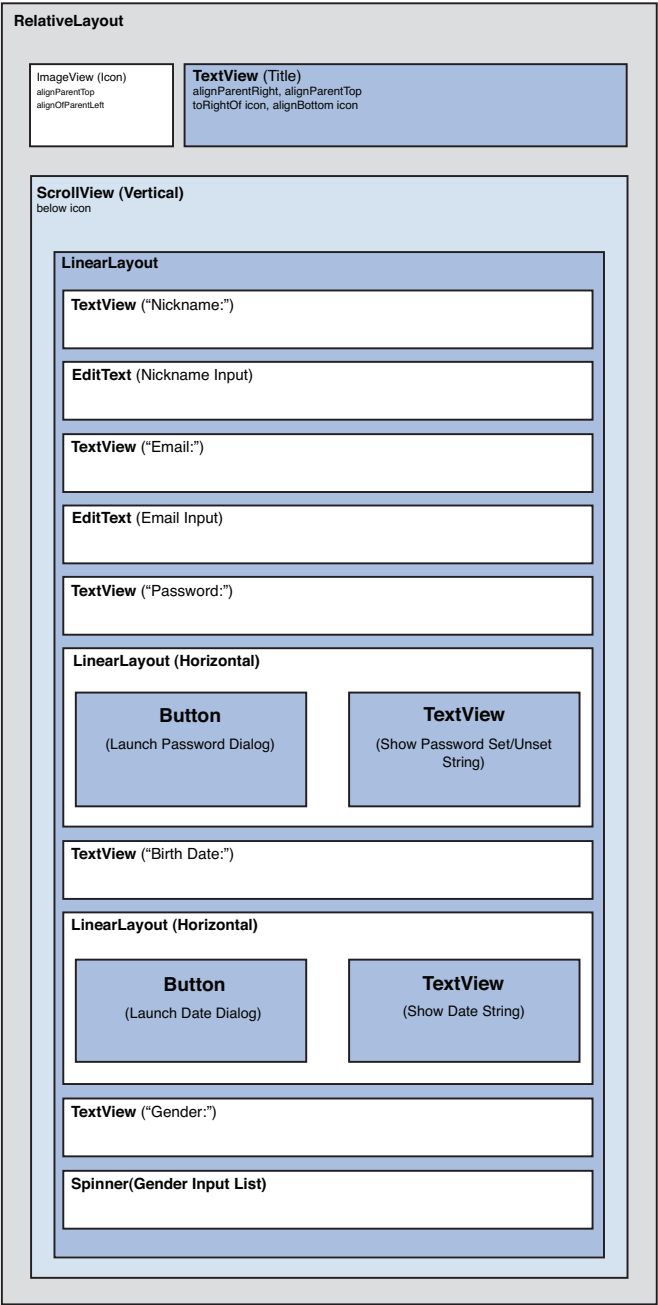


Figure 10.2 Layout Design for the Have You Read That? Settings Screen

Adding New Color Resources

The settings screen relies on one new color resource to display error text. This text color would be used when the two passwords do not match. Therefore, add the following color (#F00, which is pure red) resource to the `colors.xml` resource file:

```
<color name="error_color">#F00</color>
```

Now save the `colors.xml` resource file.

Adding New String Resources

The settings screen relies on numerous new string resources. Add the following text resources to the `strings.xml` resource file:

- Text label for each setting's `TextView` control (for example, "Nickname:")
- Text label for each `Button` control (for example, "Set Password")
- Text to display in a `TextView` control when the password is set or not set
- Text to display in a `TextView` control when the Date of Birth field is not set
- Text to display in a `TextView` control when the two Password fields match or don't match
- Text for each Gender option in the `Spinner` control (for example, "Male")

Save the `strings.xml` resource file. For a complete list of the new strings required for the settings screen, download the source code from this book's websites.

Adding New String Array Resources

`Spinner` controls can use data adapters as the source for the information they display. They can also directly use arrays for static sets of information. Android resources can be grouped together as arrays. This is a convenient way to prepare simple data for use with `Spinner` controls.

To group the gender string resources ("Male", "Female", "Prefer Not To Say") together into an array, create a new resource type called a string array.

To create a string array resource, add a new resource file called `/res/values/arrays.xml`. Within this file, create a new `<string-array>` element called `genders`. Within this `<string-array>` element, add three `<item>` elements, one for each string resource.

For example, let's assume that you created the following string resources in the `strings.xml` resource file:

```

<string
    name="gender_male">Male</string>
<string
    name="gender_female">Female</string>
<string
    name="gender_neutral">Prefer Not To Say</string>

```

Within the `arrays.xml` resource file, add each string resource as an item in the `genders` string array. For example, the first item in the array (with an index of 0) would have the value `@string/gender_neutral`. The resulting `arrays.xml` resource file would then look like this:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="genders">
        <item>@string/gender_neutral</item>
        <item>@string/gender_male</item>
        <item>@string/gender_female</item>
    </string-array>
</resources>

```

Save the `arrays.xml` resource file. To load the `genders` string array resource, access it programmatically by using the `R.array.genders` resource identifier as the identifier parameter to the `createFromResource()` method of the `ArrayAdapter` class.

Updating the Settings Screen Layout

The `settings.xml` layout file dictates the user interface of the settings screen. Follow these steps to generate the settings screen layout desired, based on your earlier design:

1. Open the `settings.xml` layout resource file in the Eclipse resource designer and remove all existing controls.
2. Add the customary `RelativeLayout` control, with its `background` attribute set to `@drawable/background_paper`. Set its `layout_width` and `layout_height` attributes to `match_parent`, so that the control fills the screen. All subsequent controls will be added inside this control.
3. Add the same header you added to other screens, using the `TextView` control and `ImageView` control, but with the title string modified.
4. Below the title bar, add a `ScrollView` control to encapsulate your settings. Set its `layout_width` and `layout_height` attributes to `match_parent`.
5. Within the `ScrollView` control, add a `LinearLayout` control to encapsulate your settings. Set its `orientation` attribute to `vertical`. Set its `layout_width` attribute to `wrap_content` and `layout_height` attribute to `match_parent`. All subsequent settings controls will be added within this `LinearLayout` control.

6. Now within the `LinearLayout` control, begin adding the settings sections themselves. Start by adding a `TextView` control to display the Nickname label text. Below the `TextView` control, add an `EditText` input control. Set its `id` attribute to `nicknameEntry`, its `maxLength` attribute to 20, its `maxLines` attribute to 1, and its `inputType` attribute to `textPersonName`.
7. Add a `TextView` control to display the Email label text and then another `EditText` control below it, setting its `id` attribute to `emailEntry`, its `maxLines` attribute to 1, and its `inputType` attribute to `textEmailAddress`.
8. Now add the Password settings region of the form by adding another `TextView` control to display the Password label text. Below it, add a horizontal `LinearLayout` control (whose `layout_height` should be `wrap_content`, `layout_width` should be `match_parent`) with two child controls: a `Button` control and a `TextView` control. Configure the `Button` control with the `id` attribute `passwordButton` and the `text` attribute set to the Password button text string resource; its `layout_width` and `layout_height` attributes should be set to `wrap_content`. Configure the `TextView` control to display the password setting state string ("Password not set", for now, until we wire up the dialog).
9. At the same level as the Password setting region, add a region for the Birth Date setting. Start by adding another `TextView` control to display the Birth Date label text. Next, add another horizontal `LinearLayout` control with two controls: a `Button` control and a `TextView` control. Configure the `Button` control with the `id` attribute `birthdayButton` and the `text` attribute set to the Birth Date button text string resource. Configure the `TextView` control to display the Birth Date setting state string ("Date not set", for now, until we wire up the dialog).
10. Add one last settings region for the Gender dropdown by adding a `TextView` control to display the Gender label text. Then, add a `Spinner` control and set its `id` attribute to `genderSpinner`.
11. Optionally, adjust any text sizes, styles, colors, and dimension attributes until the screen draws as desired.

At this point, save the `settings.xml` layout file.

Using Common Form Controls

Now that the `settings.xml` layout file is complete, you need to update the `QuizSettingsActivity` class to wire up the controls and allow editing and saving of form data. Different controls are handled in different ways. We begin with `EditText` control, and then work through `Button` and `Spinner` controls.

Working with `EditText` Controls

The `EditText` control, which is derived from the `TextView` control, is used to collect textual input from the user. Figure 10.3 shows a simple `EditText` control.



Figure 10.3 An `EditText` Control for Text Input

Configuring `EditText` Controls

All the typical attributes of a `TextView` control (for example, `textColor`, `textSize`) are available to `EditText` controls. The following are some `EditText` attributes used for the settings screen:

- `inputType`—This attribute instructs the Android system about how to help the user fill in the text. Setting the `inputType` attribute of the `EditText` control for the Email field to `textEmailAddress` instructs the Android system to use the email-oriented soft keyboard (with the @ sign). The `inputType` value called `textPassword` automatically masks the user's password as it is typed. You see this in action when you create the password dialog in the next chapter.
- `minLines` and `maxLines`—These attributes restrict the number of lines of text allowed in the control.
- `maxLength`—This attribute restricts the number of characters of text allowed in the control. For example, you limited the number of characters allowed in the Nickname setting by setting the `maxLength` attribute of the Nickname setting's `EditText` control to 20.

Handling Text Input

As with a `TextView` control, you can access the text stored in an `EditText` control by using the `getText()` and `setText()` methods. For example, to extract the string typed into the `EditText` control called `EditText_Nickname`, you use the `getText()` method, as follows:

```
EditText nicknameText = (EditText) findViewById(R.id.EditText_Nickname);
String strNicknameToSave = nicknameText.getText().toString();
```

The `getText()` method returns an `Editable` object, but since we simply want its `String` value equivalent, use the `toString()` method to get the `String` representation of the inputted text.

Working with `Button` Controls

The Android platform actually supports two kinds of button controls: the basic `Button` control and the `ImageButton` control. An `ImageButton` control behaves much like a regular `Button`

control, only instead of displaying a text label, it displays a drawable graphic. The `Button` control on the Android platform is relatively straightforward, as form controls go. Generally speaking, a `Button` control is simply a clickable area of the screen, generally with a text label. Figure 10.4 shows two `Button` controls.



Figure 10.4 Button Controls

Configuring Button Controls

Many of the typical attributes of `TextView` controls, such as `textColor` and `textSize`, are available for the `Button` text label. You need two simple `Button` controls for the settings screen: one for launching the Password dialog and one for launching the date picker dialog. Configure these `Button` controls by giving each a unique identifier and setting each control's text attribute label. Also, set each `Button` control's `layout_width` and `layout_height` attributes to `wrap_content` so that each control scales appropriately, based on the text label.

By default, a `Button` control looks like a silver rectangle with slightly rounded corners. You can use various attributes to modify the look of a `Button` control. For example, you can change the shape of the button by setting the `background`, `drawableTop`, `drawableBottom`, `drawableLeft`, and `drawableRight` attributes of the `Button` control to drawable resources.

For example, try changing the look of the `Button` control called `birthdayButton` by taking the following steps in the `settings.xml` layout file:

1. Change the `background` property of the `Button` control to a different `Drawable` graphic resource.
2. Change the `drawableTop` property of the `Button` control to a different `Drawable` graphic resource.
3. Change the `drawableBottom` property of the `Button` control to a different `Drawable` graphic resource. Did you create a monster using random `Drawable` graphic resources?
4. Change the `Button` control back to the default `Button` control look and feel by removing the `background`, `drawableTop`, and `drawableBottom` properties from `birthdayButton`. Sometimes, creating silly looking things is useful to see exactly where each resource appears and what it might look like.

Handling Button Clicks

Handling button clicks is easy. First, add a method to your activity class that takes a single `View` parameter, does not return any values (`void`), and performs the desired action when the user presses the button. Then, modify the layout file with the `Button` control and place a reference to this method as the control's `onClick` attribute.

Let's make these changes for both the Pick Date button and the Set Password button. First, add two methods to the `QuizSettingsActivity` class. Name one `onPickDateButtonClick()` and name the other `onSetPasswordButtonClick()`. Both take a single parameter of type `View` and don't return any values.

You are not yet ready to implement the dialogs that will ultimately be launched when the buttons are clicked. For the moment, it makes sense to display a debug message using a `Toast`. A `Toast` is a view that pops up in the foreground to display a message for a few seconds and then disappears. The two new methods should now look like this:

```
public void onSetPasswordButtonClick(View view) {
    Toast.makeText(QuizSettingsActivity.this,
        "TODO: Launch Password Dialog", Toast.LENGTH_LONG).show();
}
// ...
public void onPickDateButtonClick(View view) {
    Toast.makeText(QuizSettingsActivity.this,
        "TODO: Launch DatePickerDialog", Toast.LENGTH_LONG).show();
}
```

Save the Java file, and then switch over to the `settings.xml` layout resource file. Modify both the Set Password button and the Pick Date button controls by adding a value for the `android:onClick` property. For the button with the id of `passwordButton`, set this value to the string `onSetPasswordButtonClick` and for the button with the id of `birthdayButton`, set this value to the string `onPickDateButtonClick`. Make this change using either the graphic layout view and the properties panel in Eclipse or by directly editing the XML file. Either way, when the change is done, the two button entries will now look like this:

```
<Button
    android:id="@+id/passwordButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onSetPasswordButtonClick"
    android:text="@string/settings_button_pwd"></Button>
<... >
<Button
    android:id="@+id/birthdayButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/settings_button_dob"
    android:onClick="onPickDateButtonClick"></Button>
```

Save the layout file and run the application. When you click one of the buttons, you see the toast message, like the one shown in Figure 10.5.

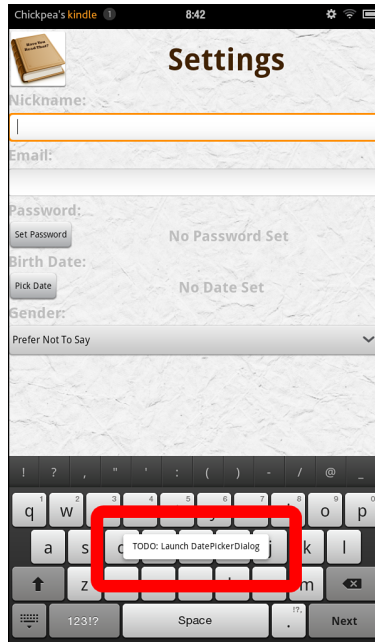


Figure 10.5 A Toast Message Triggered by a Button Click

Almost any view can actually handle button clicks. For example, back on the splash screen, we could allow the whole layout to be clickable. Simply add these two properties to the `RelativeLayout` control in the `splash.xml` file:

```
android:clickable="true"
android:onClick="onScreenTap"
```

Then, add a handler for `onScreenTap` that advances the activity to the `QuizMenuActivity` screen. Now tapping on the splash screen will advance directly the menu screen without waiting for the animations to complete.

Working with Spinner Controls

The `Spinner` control is basically the Android platform's version of a dropdown list. The `Spinner` control looks much like a dropdown when closed (see Figure 10.6, left), but when the dropdown is activated, it displays a chooser window (see Figure 10.6, right) instead of drawing the dropdown on the main screen.

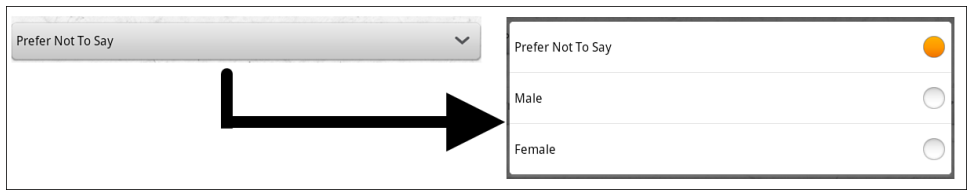


Figure 10.6 A Spinner Control Closed (Left) and Open (Right)

Configuring Spinner Controls

Earlier in this chapter, you created a string array resource of genders for the specific purpose of using it with the Spinner control. It is now time to use it.

Open up the `settings.xml` layout resource file and locate the Spinner control with the id of `genderSpinner`. Change its `entries` attribute to `@array/genders`. If you made this change in the Graphical Layout tab, you'll immediately see the first option, `Prefer Not To Say`, displayed on the preview. The XML for the Spinner control now looks like this:

```
<Spinner
    android:id="@+id/genderSpinner"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:entries="@array/genders"></Spinner>
```

Save the layout file. If you run the application now, you'll see the Spinner behave much like Figure 10.6.

Handling Spinner Selections

After the Spinner control has been filled with data, you can control which item is selected using the `setSelection()` method. For example, you know that the option for female gender is stored in the string array at index 2 because you use a 0-based string array. Because you mapped the indexes directly to the gender values, you can set the Spinner control to the Female option by using the `setSelection()` method, as follows:

```
spinner.setSelection(2);
```

The Spinner class also includes a number of methods for retrieving the current item selected.

Listening for Spinner Selection Events

You need to save the Spinner control selection to the application preferences. To do this, use the `setOnItemSelectedListener()` method of the Spinner control to register the appropriate listener for selection events. Specifically, implement the `onItemSelected()` callback method of `AdapterView.OnItemSelectedListener`, like this:

```
spinner.setOnItemSelectedListener(
    new AdapterView.OnItemSelectedListener() {
        public void onItemSelected(AdapterView<?> parent, View itemSelected,
            int selectedItemPosition, long selectedId) {
            // TODO: Save item index (selectedItemPosition) as Gender setting
        }

        // ... Other required overrides
    });
```

At this point, save the `QuizSettingsActivity.java` file and run the Have You Read That? application in the Android emulator or your Kindle Fire. You're almost done; now you need to commit the form data to the application shared preferences.

Saving Form Data with SharedPreferences

You can use the persistent storage mechanism called `SharedPreferences` to store the application game settings. Using these preferences, you can save all the form values on the settings screen.

Defining SharedPreferences Entries

Earlier, you added a string to the `QuizActivity` base class for your game preferences:

```
public static final String GAME_PREFERENCES = "GamePrefs";
```

Now add a `String` variable for each of the settings to store to the `QuizActivity` class preferences:

```
public static final String GAME_PREFERENCES_NICKNAME = "Nickname"; // String
public static final String GAME_PREFERENCES_EMAIL = "Email"; // String
public static final String GAME_PREFERENCES_PASSWORD = "Password"; // String
public static final String GAME_PREFERENCES_DOB = "DOB"; // Long
public static final String GAME_PREFERENCES_GENDER = "Gender"; // Int
```

Saving Settings to SharedPreferences

Now that you have defined the preference settings, it's time to save the form fields to the preferences. Within the `QuizSettingsActivity` class, begin by defining a `SharedPreferences` member variable:

```
private SharedPreferences mGameSettings;
```

Within the `onCreate()` method of the activity, initialize this member variable as follows:

```
mGameSettings = getSharedPreferences(GAME_PREFERENCES, Context.MODE_PRIVATE);
```

Pass in the name of your `SharedPreferences` (the `String` called `GAME_PREFERENCES` found in the `QuizActivity` class). The mode called `MODE_PRIVATE` is the default permission used for private application files.

Now, any time you need to save a preference within your application, simply open a `SharedPreferences.Editor`, assign a specific preference setting, and commit the change. For example, to save the Nickname `EditText` information, retrieve the text by using the `EditText` control's `getText()` method:

```
final EditText nicknameText =
    (EditText) findViewById(R.id.nicknameEntry);
String strNickname = nicknameText.getText().toString();
```

After you extract the `String` value from the `EditText` input field, save it to `SharedPreferences.Editor`, using the `putString()` method:

```
Editor editor = mGameSettings.edit();
editor.putString(GAME_PREFERENCES_NICKNAME, strNickname);
editor.commit();
```

The Nickname, Email, and Password settings are saved as string values, but the Birth Date and Gender settings are of long and integer types, respectively. To save these settings, extract the value from the appropriate control, convert it if necessary, and save it using the `SharedPreferences.Editor` methods `putLong()` and `putInt()`.

For now, commit the input from the Nickname, Email, and Gender fields. Add the Nickname and Email commit code to the `onPause()` method of the `QuizSettingsActivity` class. (You may need to create the method stub first, if you haven't already.) The Gender setting can be saved within the `Spinner` listener that you implemented earlier.

Further work needs to be done with the Date of Birth and Password fields before you can collect the user's input and save it to the settings. You revisit this in the next chapter, when you implement the Pick Date and Set Password dialogs.

Reading Settings from `SharedPreferences`

When you begin saving settings in a persistent fashion, you need to be able to read them back out and load them into the form for editing. To do this, access the game preferences and check whether specific settings exist. Do this for the Nickname setting by using the `contains()` and `getString()` methods of `SharedPreferences` as follows:

```
final EditText nicknameText =
    (EditText) findViewById(R.id.EditText_Nickname);
if (mGameSettings.contains(GAME_PREFERENCES_NICKNAME)) {
    nicknameText.setText(mGameSettings.getString(
        GAME_PREFERENCES_NICKNAME, ""));
}
```

This code first checks for the existence of a specific setting name defined as `GAME_PREFERENCES_NICKNAME` in the shared preferences by using the `contains()` method. If the `contains()` method returns `true`, extract the value of that setting (a `String` setting) from shared preferences by using the `getString()` method.

The Nickname, Email, and Password settings are strings and can be extracted using the `getString()` method. However, the Birth Date setting must be extracted using the `getLong()` method, and the Gender setting requires the `getInt()` method.

Finally, for testing purposes, consider overriding the `onDestroy()` method of `QuizSettingsActivity` to log all current settings whenever the settings screen is destroyed:

```
@Override
protected void onDestroy() {
    Log.d(DEBUG_TAG, "SHARED PREFERENCES");
    Log.d(DEBUG_TAG, "Nickname is: "
        + mGameSettings.getString(GAME_PREFERENCES_NICKNAME, "Not set"));
    Log.d(DEBUG_TAG, "Email is: "
        + mGameSettings.getString(GAME_PREFERENCES_EMAIL, "Not set"));
    Log.d(DEBUG_TAG, "Gender (M=1, F=2, U=0) is: "
        + mGameSettings.getInt(GAME_PREFERENCES_GENDER, 0));
    // We are not saving the password yet
    Log.d(DEBUG_TAG, "Password is: "
        + mGameSettings.getString(GAME_PREFERENCES_PASSWORD, "Not set"));
    // We are not saving the date of birth yet
    Log.d(DEBUG_TAG, "DOB is: "
        + DateFormat.format("MMMM dd, yyyy", mGameSettings.getLong(
            GAME_PREFERENCES_DOB, 0)));
    super.onDestroy();
}
```

Now whenever `QuizSettingsActivity` is destroyed (for example, when a user presses the Back button), the preferences that have been committed are displayed in the LogCat console. Once the Pick Date and Set Password dialogs are functioning, you'll see the correct debug information here. Because the `onDestroyed()` method is called after `onPause()`, all changes to settings will be reflected.

Summary

In this chapter, you added a form to the settings screen of the Have You Read That? trivia application. The form handles various fields, including text input of various kinds, using `EditText` controls, and a dropdown list, using a `Spinner` control. You also conserved screen space by implementing two `Button` controls, which can be wired up in the future to launch dialogs in the next chapter. Finally, you implemented a simple `SharedPreferences` mechanism to load and save game settings for use in the application.

Exercises

1. Add a `Toast` to the gender `Spinner` control listener. Make it display the new value when the preferences are successfully saved, confirming that the value was saved. Think about if this is useful feedback to the user or not.
2. Implement a `Clear` button that, when clicked, resets or deletes all game preferences using the `clear()` method of `SharedPreferences.Editor`. Don't forget to call the `commit()` method to save your changes to the preferences after you clear them.
3. [Challenging!] Modify each `EditText` control to save its contents when the user presses the enter key (`KEYCODE_ENTER`). Hint: Use an `OnKeyListener` with the `EditText` controls. Look up how it works in the Android reference documentation. It will be similar in style to the `Spinner` listener.
4. [Challenging!] Experiment with the `PreferenceActivity` class discussed in a tip near the beginning of this chapter. Create an alternative settings activity class and try to design it using the `PreferenceActivity` class. How does this method differ from the method shown in this chapter? Which method do you prefer? There is sample code provided in the class documentation to get you going if you have trouble.
5. [Challenging!] The settings layout file is complex and therefore not necessarily as efficient as it could be. Could you redesign the layout to make use of a single `RelativeLayout` control instead of the nested `LinearLayout` controls within the `ScrollView`? Give it a shot!

Using Dialogs to Collect User Input

In this chapter, you continue to add features to the Have You Read That? settings screen. Specifically, you learn about Android activity dialogs and implement several within the `QuizSettingsActivity` class. Each dialog will be specially designed to collect a specific type of input from the user. First, you implement a `DatePickerDialog` to collect the user's birth date, and then you build a custom dialog to enable the user to change her password.

Working with Activity Dialogs

There is only so much screen real estate available on a device, but it's cumbersome—both for the developer and the user alike—to have to transition between activities too frequently for simple tasks. Luckily, the Android SDK includes a concept called a *dialog*. An `Activity` class can use `Dialog` classes of various types to organize information and react to user-driven events without having to spawn full sub-activities. For example, an activity might display a `Dialog` informing the user of an error or asking to confirm an action, such as deleting a piece of information. Using the `Dialog` mechanism for simple tasks helps keep the number of `Activity` classes within an application manageable.

Deprecated Dialog Calls

If you're looking up the method calls we introduce in this chapter for `Activity` dialogs, you may notice that the SDK documentation considers them deprecated. The suggestion is to use `DialogFragment` with the `FragmentManager` instead. This is only available on Android API Level 11 and higher or using the Android Support Library. While the Android Support Library can be used with Kindle Fire, these calls weren't deprecated in API Level 10 and will continue to work without issue.

Exploring the Different Types of Dialogs

A number of different `Dialog` types are available in the Android SDK, including the following:

- **Dialog**—The basic class for all dialog types (see Figure 11.1A). The simplest type of dialog, this control can be used to inform the user.
- **AlertDialog**—A dialog with one, two, or three `Button` controls (see Figure 11.1B). This type of dialog is often used to get user confirmation (or denial) of an operation. For example, you can use it to confirm the deletion of a file.
- **CharacterPickerDialog**—A dialog for choosing an accented character associated with a base character (see Figure 11.1C). This type of dialog is often used to provide a subset of characters to the user for selection.
- **DatePickerDialog**—A dialog with a `DatePicker` control (see Figure 11.1D). This type of dialog is used to collect date input from the user.
- **ProgressDialog**—A dialog with a determinate or indeterminate `ProgressBar` control (see Figure 11.1E). This type of dialog is used to inform the user about the status, or progress, of an operation. For example, it is used to inform the user that data is being transferred to or from the network.
- **TimePickerDialog**—A dialog with a `TimePicker` control (see Figure 11.1F). This type of dialog is used to collect time input from the user.

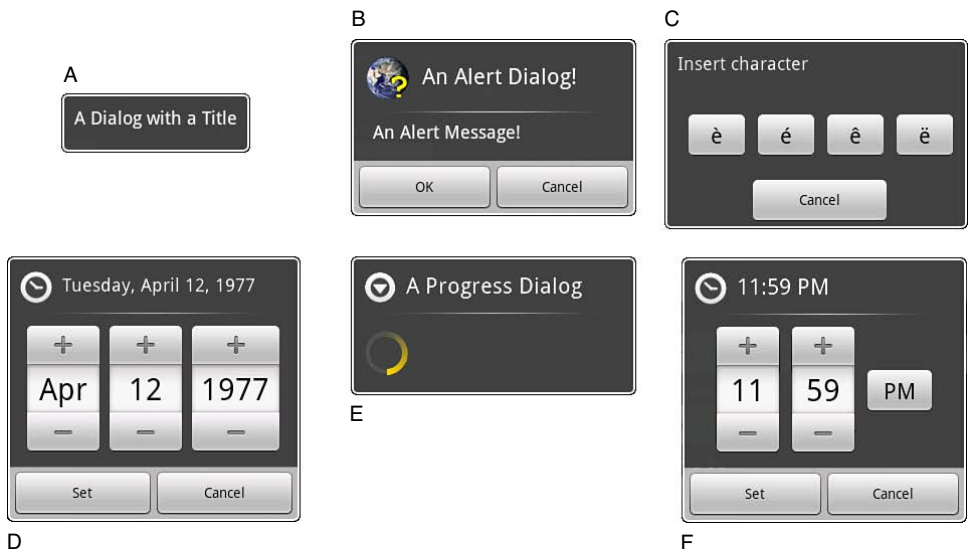


Figure 11.1 Different `Dialog` Types Available in Android

If none of the existing dialog types is adequate for your needs, you can create custom dialogs that meet your specific layout requirements using a dialog builder. We discuss custom dialogs later in this chapter when we implement the password dialog.

Tracing the Lifecycle of a Dialog

Each `Dialog` instance must be defined within the activity in which it will be used; the dialog is exclusive to that `Activity` class. A dialog may be launched once or used repeatedly. Understanding how an activity manages the dialog lifecycle is important to correctly implement a dialog. Let's look at the key methods that an activity must use to manage a `Dialog` instance:

- The `showDialog()` method is used to display a `Dialog` instance.
- The `dismissDialog()` method is used to stop showing a `Dialog` instance. The `Dialog` is kept around in the activity's dialog pool. If the `Dialog` is shown again using the `showDialog()` method, then the cached version will be displayed again.
- The `removeDialog()` method is used to remove a `Dialog` instance from the `Activity` object's dialog pool. The `Dialog` instance will no longer be kept around for future use. If you call the `showDialog()` method again, the `Dialog` must be re-created.

Defining a Dialog

Dialogs must be defined in advance. Each `Dialog` instance must have a unique dialog identifier (an integer that you define) associated with it. You must override the `onCreateDialog()` method of the `Activity` class and have it return the appropriate `Dialog` instance for the given identifier. If the activity has multiple dialogs, the `onCreateDialog()` method can use a `switch` statement to return the appropriate `Dialog`, based on the incoming parameter—the dialog identifier.

Initializing a Dialog

Because a `Dialog` instance may be kept around by an activity, it can sometimes be important to reinitialize or refresh the dialog each time it is shown instead of just when it is created the first time. We say sometimes because some dialogs do not require refreshes; for example, a static dialog that just shows a text label with some buttons would not need to be refreshed, whereas a complex dialog with input controls or progress bars would likely need to be reset. If you need a dialog to be reinitialized each time it is shown, you can override the `onPrepareDialog()` method of the `Activity` class and alter the dialog's contents.

Although the `onCreateDialog()` method may be called only once for initial dialog creation, the `onPrepareDialog()` method is called each time the `showDialog()` method is called, giving the activity a chance to initialize the dialog each time it is shown to the user.

Launching a Dialog

Any `Dialog` instance already defined within an activity is shown by calling the `showDialog()` method and passing in a valid dialog identifier—in other words, one that will be recognized by the `onCreateDialog()` method.

Dismissing a Dialog

Most dialogs have automatic dismissal circumstances in the form of `Button` controls. However, to force a dialog to be dismissed programmatically, simply call the `dismissDialog()` method and pass in the appropriate dialog identifier.

Removing a Dialog from Use

Dismissing a dialog does not destroy it or remove it from the activity's dialog pool. If the dialog is shown again using the `showDialog()` method, its cached contents will be redisplayed. To force an activity to remove a `Dialog` instance from its pool and not reuse it, call the `removeDialog()` method and pass in the valid dialog identifier.

Using the `DatePickerDialog` Class

Let's turn our attention to implementing a proper dialog on the settings screen of the *Have You Read That?* application. We start with a simple dialog to collect the user's date of birth. To achieve this feature, you must add a `DatePickerDialog` to the `QuizSettingsActivity` class, which involves several steps:

1. Defining a unique identifier for the dialog within the `QuizSettingsActivity` class.
2. Implementing the `onCreateDialog()` method of the `Activity` class to create and return a `DatePickerDialog` instance when supplied the proper unique identifier.
3. Implementing the `onPrepareDialog()` method of the activity to initialize `DatePickerDialog` with the birth date preference or the current date. You can use the `Calendar` class to get the current date on the device. The `Calendar` class has fields for each of the "parts" of the date: day, month, and year. You can use this feature of the `Calendar` class to configure `DatePickerDialog` with a specific date.
4. Updating the `Pick Date` Button control's click handler (called `onPickDateButtonClick()` in the sample code) to launch the `DatePickerDialog` using the `showDialog()` method, with the unique dialog identifier.

Now that you know what steps to take to create your first dialog, let's walk through them individually.

Adding a `DatePickerDialog` to a Class

To create a `DatePickerDialog` instance within the `QuizSettingsActivity` class, first define a unique identifier to represent the dialog for the class, as follows:

```
private static final int DATE_DIALOG_ID = 101;
```

Next, implement the `onCreateDialog()` method of the `QuizSettingsActivity` class and include a `switch` statement with a `case` statement for the new dialog identifier, like this:

```
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case DATE_DIALOG_ID:
            // TODO: Return a DatePickerDialog here
    }
    return null;
}
```

Now let's look at how to construct a `DatePickerDialog` instance. Within the `case` statement for `DATE_DIALOG_ID`, you must return a valid `DatePickerDialog` instance. The constructor for the `DatePickerDialog` class includes a `DatePickerDialog.OnDateSetListener` parameter. This parameter can be used to provide an implementation of the `onDateSet()` method to handle when the user chooses a specific date within the picker. Use this method to save the date to the `SharedPreferences`, like this:

```
Calendar now = Calendar.getInstance();
DatePickerDialog dateDialog =
    new DatePickerDialog(this,
        new DatePickerDialog.OnDateSetListener() {
            public void onDateSet(DatePicker view, int year,
                int monthOfYear, int dayOfMonth) {
                Time dateOfBirth = new Time();
                dateOfBirth.set(dayOfMonth, monthOfYear, year);
                long dtDob = dateOfBirth.toMillis(true);
                dobInfo.setText(DateFormat
                    .format("MMMM dd, yyyy", dtDob));
                Editor editor = mGameSettings.edit();
                editor.putLong(GAME_PREFERENCES_DOB, dtDob);
                editor.commit();
            }
        }, now.get(Calendar.YEAR), now.get(Calendar.MONTH),
        now.get(Calendar.DAY_OF_MONTH));
```

A `DatePicker` control has three different input controls: a month picker, a day picker, and a year picker. Therefore, to create a valid instance of a `DatePickerDialog`, you must set these values individually. Because the `DatePickerDialog` can be launched any number of times, do not initialize the picker date information within the `onCreateDialog()` method. Instead, pass in default values (such as today's year, month, and day from the `Calendar` class). Then, set the values to display in the `onPrepareDialog()` method. Once you have a valid `DatePickerDialog` instance, return it:

```
return dateDialog;
```

Initializing a DatePickerDialog

To initialize the `DatePickerDialog` each and every time it is displayed, not just when it is first created, override the activity's `onPrepareDialog()` method to set the `DatePicker` control's month, day, and year values to either today's date or the user's birth date as it is saved in the current game preferences.

The `onPrepareDialog()` method receives both the dialog identifier and the specific instance of the `Dialog` in order to modify the related instance, as needed. To update the date values of `DatePickerDialog`, use the `updateDate()` method as shown in this implementation of the `onPrepareDialog()` method:

```
@Override
protected void onPrepareDialog(int id, Dialog dialog) {
    super.onPrepareDialog(id, dialog);
    switch (id) {
        case DATE_DIALOG_ID:
            // Handle any DatePickerDialog initialization here
            DatePickerDialog dateDialog = (DatePickerDialog) dialog;
            int day, month, year;
            // Check for date of birth preference
            if (mGameSettings.contains(GAME_PREFERENCES_DOB)) {
                // Retrieve Birth date setting from preferences
                long msBirthDate = mGameSettings.getLong(
                    GAME_PREFERENCES_DOB, 0);
                Time dateOfBirth = new Time();
                dateOfBirth.set(msBirthDate);
                day = dateOfBirth.monthDay;
                month = dateOfBirth.month;
                year = dateOfBirth.year;
            } else {
                Calendar now = Calendar.getInstance();
                // Today's date fields
                day = now.get(Calendar.DAY_OF_MONTH);
                month = now.get(Calendar.MONTH);
                year = now.get(Calendar.YEAR);
            }
            // Set the date in the DatePicker to the date of birth OR to the
            // current date
            dateDialog.updateDate(year, month, day);
            return;
    }
}
```

Launching DatePickerDialog

You configured `DatePickerDialog`, but it doesn't display unless the user clicks the appropriate Pick Date Button control on the main settings screen. The user triggers `DatePickerDialog` by

pressing the Button control called `birthdayButton`. This triggers the `onPickDateButtonClick()` method previously implemented.

Replace the Toast message within the button handler. Call the `showDialog()` method instead, which launches `DatePickerDialog`, as shown in Figure 11.2:

```
public void onPickDateButtonClick(View view) {
    showDialog(DATE_DIALOG_ID);
}
```

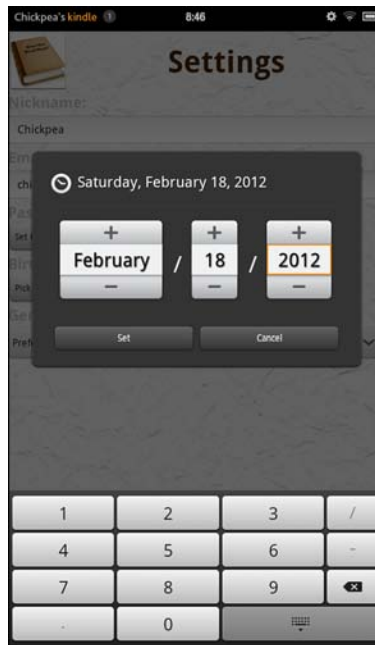


Figure 11.2 `DatePickerDialog` Used for Birth Date Input

Voilà! You've completed your first dialog. Now turn your attention to the more complex password dialog.

Working with Custom Dialogs

When the basic dialog types do not suit your purpose, you can create a custom dialog. To create a custom dialog, begin with an `AlertDialog` instance and use an `AlertDialog.Builder` class to override its default layout and provide alternative functionality. To create a custom dialog this way, follow these steps:

1. Design a custom layout resource to display in `AlertDialog`.
2. Define the custom dialog identifier in the activity.
3. Update the `Activity` class's `onCreateDialog()` method to build and return the appropriate custom `AlertDialog`.
4. Launch the dialog using the `showDialog()` method.

Adding a Custom Dialog to the Settings Screen

The Have You Read That? setting screen requires a password confirmation dialog. However, this type of dialog is not available within the Android SDK, so you need to create a custom dialog to provide this functionality. Figure 11.3 shows how a password dialog might behave when passwords match or don't match.

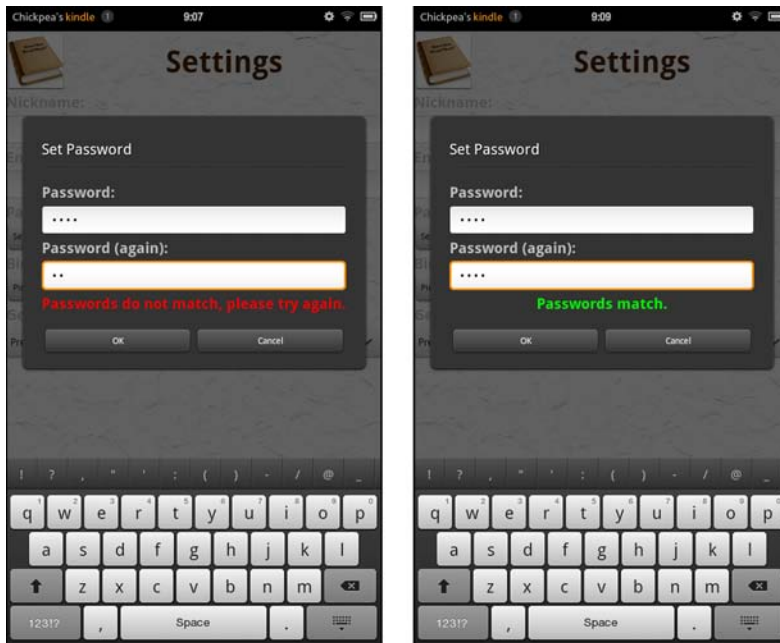


Figure 11.3 Simple Password Dialog Behavior

The custom dialog requires two text input fields for entering password data. When the two passwords match, the password will be set. Figure 11.4 shows a rough design of the settings screen in this case.

Dialog Title

PASSWORD:
(Text Hidden as Typed)

PASSWORD (Again):
(Text Hidden as Typed)

“Passwords Match”

OK Cancel

Figure 11.4 A Custom Dialog Used for Handling Password Input

The password dialog is simply a subform of the settings screen that contains two `EditText` input fields as well as a `TextView` control below the input fields to inform the user in real-time whether or not the passwords match.

Figure 11.5 shows the layout design of the password dialog.

LinearLayout (Vertical Orientation)

TextView (“Password:”)

EditText (Password #1 Input)

TextView (“Password (Again):”)

EditText (Password #2 Input)

TextView (e.g. “Passwords Match”)

Figure 11.5 Layout Design for the Have You Read That? Settings Screen

You can take advantage of the built-in `Button` controls that can be configured for use with `AlertDialog`. The three (or fewer) buttons need not be included in your custom dialog layout design.

Implementing the Password Dialog Layout

Now it's time to implement the new layout resource for the password dialog. Begin by creating a new layout resource file called `password_dialog.xml`. This layout resource file represents the contents of the dialog. To create this file, follow these steps:

1. Open the Eclipse layout resource editor and add a new resource file called `/res/layout/password_dialog.xml` to your project.
2. Add a `LinearLayout` control. Set its `orientation` attribute to `vertical`. Set its `layout_width` and `layout_height` attributes to `match_parent`. All subsequent controls will be added inside this `LinearLayout` control.
3. Add a `TextView` control to display the Password label text. Then, add an `EditText` control and set its `id` attribute to `passwordEntry`, its `maxLength` attribute to 20, and its `inputType` attribute to `textPassword`.
4. Add another `TextView` control to display the Password label text again. Then, add another `EditText` control and set its `id` attribute to `passwordConfirmEntry`, its `maxLength` attribute to 20, and its `inputType` attribute to `textPassword`.
5. Add a `TextView` control with the `id` attribute `passwordEntryStatus` to display the password status label text. This `TextView` control will display whether the two password fields match in real time.
6. Finally, modify any of the controls' attributes, like colors, styles, and text sizes, to suit your tastes.

At this point, save the `password_dialog.xml` layout file.

Adding the Password Dialog to an Activity Class

To add a custom `AlertDialog` to the `QuizSettingsActivity` class, you must first declare a unique identifier to represent the dialog, as follows:

```
private static final int PASSWORD_DIALOG_ID = 102;
```

Next, update the `onCreateDialog()` method of the `QuizSettingsActivity` class to include a case statement for the new dialog identifier:

```
case PASSWORD_DIALOG_ID:
    // Build Dialog
    // Return Dialog
```

Now, let's look at how to build the custom password dialog from the ground up. Begin by inflating (loading) the custom layout you created into a `View` control:

```
LayoutInflater inflater =
    (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
final View layout =
    inflater.inflate(R.layout.password_dialog, null);
```

To load the `password_dialog.xml` layout file into a `View` object, you must retrieve the `LayoutInflater` and then call its `inflate()` method, passing in the layout resource identifier and null for the parent as the system will control assigning the layout into a hierarchy.

Once the custom layout has been inflated into a `View`, it can be acted upon programmatically, much like a regular layout. At this point, controls can be populated with data and event listeners can be registered. For example, to retrieve the `EditText` and `TextView` controls from the `view` instance called `layout`, use the `findViewById()` method for that `view` control (as opposed to the `Activity` as a whole), as follows:

```
final EditText password = (EditText) layout
    .findViewById(R.id.passwordEntry);
final EditText passwordConfirm = (EditText) layout
    .findViewById(R.id.passwordConfirmEntry);
final TextView passwordStatus = (TextView) layout
    .findViewById(R.id.passwordEntryStatus);
```

At this point, you can register any event listeners on the `EditText` fields, such as those discussed earlier to watch `EditText` input and match the strings as the user types.

Listening for `EditText` Keystrokes

When working with `EditText` controls, you can listen for keystroke events while the user is still typing. For example, you can check the text strings within two `EditText` password fields while the user is typing and report if they match or not. A third `TextView` control, called `passwordEntryStatus`, provides “live” feedback about whether the passwords match.

First, register a `TextWatcher` with the second `EditText` control, using the `addTextChangedListener()` method, like this:

```
final TextView passwordStatus = (TextView) layout
    .findViewById(R.id.passwordEntryStatus);

passwordConfirm.addTextChangedListener(new TextWatcher() {
    @Override
    public void afterTextChanged(Editable s) {
        String passwordString = password.getText().toString();
        String confirmString = passwordConfirm.getText().toString();
        if (passwordString.equals(confirmString)) {
            passwordStatus.setText(R.string.settings_pwd_equal);
            passwordStatus.setTextColor(getResources().getColor(
                R.color.good_color));
        } else {
            passwordStatus.setText(R.string.settings_pwd_not_equal);
            passwordStatus.setTextColor(getResources().getColor(
                R.color.error_color));
        }
    }
})
```

```
// ... other overrides do nothing

});
```

The `TextWatcher` has a number of methods that require implementation. However, the one we're interested in is the `afterTextChanged()` method. Now, the user can type the password into the `password` `EditText` control normally. However, each time the user types a character into the `passwordConfirm` `EditText` control, the text is compared to the text in the first `EditText` control and the text of the `TextView` control called `passwordStatus` is updated to reflect whether the text matches.

Now that you have inflated the layout into a `View` object and configured it for use, you can attach it to `AlertDialog`. To do this, use the `AlertDialog.Builder` class:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setView(layout);
builder.setTitle(R.string.settings_button_pwd);
```

First, set the view of `AlertDialog.Builder` to the inflated layout using the `setView()` method. Then set the title of the dialog with the `setTitle()` method (Set Password).

This dialog will have two `Button` controls: a positive button (OK) and a negative button (Cancel). Because you do not want this dialog cached for reuse by the activity (which would cache typed-in password contents in the `EditText` controls), both `Button` handlers should call the `removeDialog()` method, which destroys the dialog:

```
QuizSettingsActivity.this.removeDialog(PASSWORD_DIALOG_ID);
```

The positive button (OK) requires some additional handling. When the user clicks this button, extract the password text from the `EditText` controls, compare the results, and, if two strings match, store the new password in the shared preferences of the application. Configure the positive button using the `setPositiveButton()` method of the builder, like this:

```
builder.setPositiveButton(android.R.string.ok,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,
            int which) {
            TextView passwordInfo = (TextView) findViewById(R.id.passwordInfo);
            String passwordString = password.getText()
                .toString();
            String confirmString = passwordConfirm
                .getText().toString();
            if (passwordString.equals(confirmString)) {
                Editor editor = mGameSettings.edit();
                editor.putString(
                    GAME_PREFERENCES_PASSWORD,
                    passwordString);
            }
        }
    });
```

```

        editor.commit();
        passwordInfo
            .setText(R.string.settings_pwd_set);
    } else {
        Log.w(DEBUG_TAG,
            "Warning: Passwords do not match." +
            "Not saving. Keeping old password (if set).");
    }
    QuizSettingsActivity.this
        .removeDialog(PASSWORD_DIALOG_ID);
    }
});

```

The negative button (Cancel) simply returns the user to the main screen. Configure the negative button using the `setNegativeButton()` method of the builder, like this:

```

builder.setNegativeButton(android.R.string.cancel,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            QuizSettingsActivity.this
                .removeDialog(PASSWORD_DIALOG_ID);
        }
    });

```

When your dialog is fully configured using the builder, you call its `create()` method to generate the custom `AlertDialog` and return it:

```

AlertDialog passwordDialog = builder.create();
return passwordDialog;

```

Launching the Custom Password Dialog

A custom dialog, such as your password dialog, is launched the same way as a regular dialog: using the `showDialog()` method of the activity. On the settings screen of the Have You Read That? application, the user triggers the custom password dialog to launch by pressing the Button control called `passwordButton`. Therefore, you can update this control's click handler (called `onSetPasswordButtonClick()` in the sample source code) to launch the password dialog accordingly:

```

public void onSetPasswordButtonClick(View view) {
    showDialog(PASSWORD_DIALOG_ID);
}

```

Figure 11.6 shows the resulting settings screen, with dialog controls.

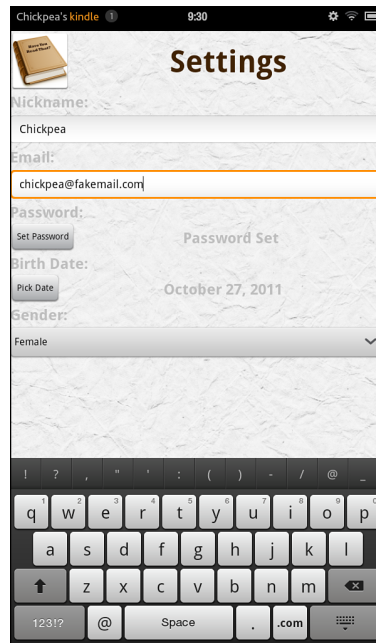


Figure 11.6 The Complete Have You Read That? Settings Screen

Summary

In this chapter, you learned how an activity can use dialog controls to simplify screen functionality and layout—specifically on the settings screen of the Have You Read That? application. A dialog can be used to display appropriate information to the user in the form of a pop-up window. There are dialog types for inputting dates, times, and special characters, as well as helper types for showing progress or displaying alert messages. You can also create custom dialog controls.

Exercises

1. Add another dialog control to the screen. Try using a different type of dialog than those already used. What other information might you want to collect from the user that is well suited for a dialog?
2. Update the custom password dialog to display the number of characters typed alongside the password status.

3. Update the custom password dialog and change the colors of the status text in the `TextView` control when passwords match (white) or do not match (blue).
4. [Challenging!] Experiment with the `onPrepareDialog()` method. Try moving the password initialization code to the `onCreateDialog()` method and note how the dialog behaves differently and caches the contents of the controls when the dialog is launched multiple times.

This page intentionally left blank

Adding Application Logic

In this chapter, you wire up the screen at the heart of the Have You Read That? application: the game play screen. This screen prompts the user to answer a series of books and stores the resulting score information. Because the screen must display a dynamic series of images and text strings, you will leverage several new `View` controls, including `ImageSwitcher` and `TextSwitcher` controls, to help transition between books in the game. You will also need to update the `QuizGameActivity` class with game logic and game state information, including the retrieval of batches of new books, as a user progresses through the books.

Numerous intermediate Java topics are referenced in this chapter, including the use of inner classes, class factories, and data structure classes, like the `Hashtable` class. Although these are not Android topics per se, we provide some light explanation when we use these features. That said, readers are expected to either be familiar with such topics or be willing to look them up in the Java reference of their choice.

Designing the Game Screen

The game screen leads the user through a series of books and logs the number of positive responses as the score. Each book has text and a corresponding graphic to display. For example, the game screen might display a picture of a book, ask if the user has ever read this book, and record one of two responses: Yes or No.

Unlike the screens you developed in previous chapters, the game screen does not require the customary title bar. Instead, you will want to use the entire screen to display the game components. Figure 12.1 shows a rough design of the game screen.

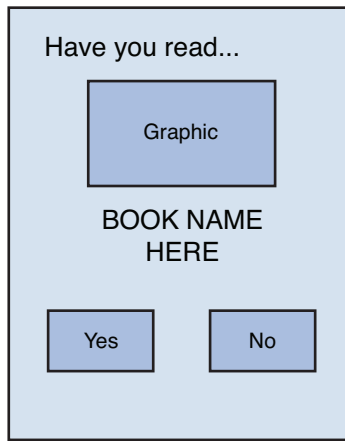


Figure 12.1 Rough Design for the Have You Read That? Game Screen

Despite the lack of header, you still want the game screen to share some common features with the rest of the application: It should use the same background graphic, font, and color scheme as the other screens. To translate this rough design into the appropriate layout design, update the `/res/layout/game.xml` layout file and the `QuizGameActivity` class.

The `RelativeLayout` control works especially well for displaying items, such as `Button` controls, in the bottom corners of the screen. You can also use a vertically oriented `LinearLayout` control to display the text (in a `TextView` control) and graphic (in an `ImageView` control) related to each book to the user, or you can arrange everything within the `RelativeLayout` control.

Buttons will be used to handle the user responses and drive the application. Each time the user clicks a `Button` control, the game screen updates the `ImageView` and `TextView` controls to display the next book. To smoothly transition (and animate) from one book to the next, you can use the special view controls `ImageSwitcher` and `TextSwitcher`, which are subclasses of the `ViewSwitcher` class (`android.widget.ViewSwitcher`).

A `ViewSwitcher` control can animate between two child view controls: the current view control and the next view control to display. Only one view control is displayed at any time, but animations, such as fades or rotates, can be used during the transition between view controls. These child view controls are generated using the `ViewFactory` class. For example, `ImageSwitcher` and its corresponding `ViewFactory` can be used to generate the current book `ImageView` and switch in the next book's `ImageView` when the user clicks a `Button` control. Similarly, a `TextSwitcher` control has two child `TextView` controls, with transitional animation applying to the text.

Figure 12.2 shows the layout design of the game screen, which uses an `ImageSwitcher` control and a `TextSwitcher` control. Each time the user clicks a `Button` control, the two switcher controls generate a new `TextView` and `ImageView` to display on the screen with the data for the next book.

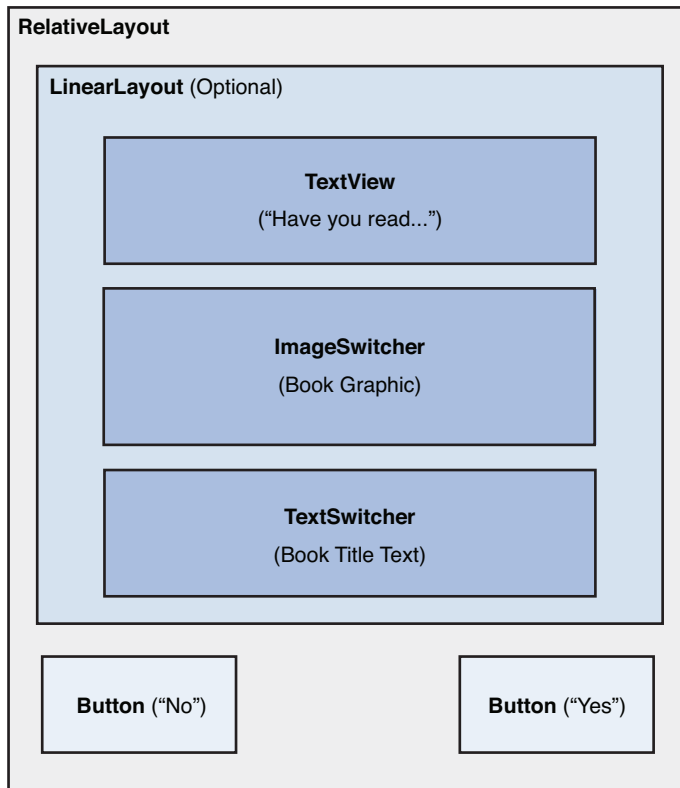


Figure 12.2 Layout Design for the Have You Read That? Game Screen with `ImageSwitcher` and `TextSwitcher` Controls

Implementing the Game Screen Layout

To implement the game screen, begin by adding numerous new resources to the project. Then, update the `game.xml` layout resource to reflect the game screen design. Let's do this now.

Adding New Project Resources

For the game screen, add some new resources:

- String resources to display on Button controls and error messages
- Dimension resources needed to display the game screen controls
- Drawable resource for when no book is available to display
- Two XML resources with mock book batches

Adding New String Resources

The game screen relies on numerous new string resources. Add the following text resources to the `strings.xml` resource file:

- Text labels for each Button control (for example, Yes and No)
- Text to display when no books are available

Save the `strings.xml` resource file. For a complete list of the new strings required for the game screen, see the sample source code provided for this chapter on this book's websites.

Adding New Dimension Resources

The game screen relies on several new dimension resources. These dimensions are used to size the Button controls and display the book data. Therefore, add the following dimension resources to the `dimens.xml` resource file:

```
<dimen name="game_book_title_text_size">32dp</dimen>
<dimen name="game_book_header_text_size">60dp</dimen>
<dimen name="game_button_text_size">40dp</dimen>
<dimen name="game_button_padding_size">25dp</dimen>
```

Now save the `dimens.xml` resource file.

Adding New XML Resources

Eventually, the books used by the Have You Read That? application will be retrieved from a server on the Internet as a chunk of XML data. XML is one of the most common mechanisms for transferring structured data, so we use it to store game book data, much as we did with the score data.

Just as we did for the score data, begin by creating two batches of mock books that can be accessed locally as XML: `/res/xml/mock_booklist_1.xml` and `/res/xml/mock_booklist_2.xml`.

In Chapter 13, “Adding Network Support,” when you add network support to the application, you will retrieve XML content in this same structure from a remote server. However, by

including mock batches of books for now, you have the opportunity to iron out the game logic without worrying about network connectivity, threading, or other more advanced topics that cannot be crammed into a one-chapter lesson.

Regardless of whether the XML book batches are sourced locally or from a remote server, the XML content is the same. Here is what it looks like:

```
<?xml version="1.0" encoding="utf-8"?>
  <!-- This is a mock book XML chunk -->
<books>
  <book
    number="1"
    title=
      "The Great Gatsby?"
    coverUrl=
      "http://commondatastorage.googleapis.com/kf6nvr
➡/hyreadthat/gatspy.png"
  />
  <book
    number="2"
    title=
      "The Catcher in the Rye?"
    coverUrl=
      "http://commondatastorage.googleapis.com/kf6nvr/hyreadthat/rye.png"
  />
</books>
```

As you can see, the XML is very simple. It has one tag called `<books>`, which can contain any number of `<book>` tags. Each `<book>` tag has three attributes: the book identifier (`number`), the book title (`title`), and the URL to the image associated with the book (`coverUrl`). The images are graphics sourced from a public server. Instead of adding each and every book graphic to the resources of the application, this saves time now because the images will ultimately come from a server.

Updating the Game Screen Layout

The `game.xml` layout file dictates the user interface of the game screen. The Eclipse layout resource editor does not display `TextSwitcher` or `ImageSwitcher` controls in design mode, so you may want to work in XML mode. Follow these steps to generate the layout you want, based on your design:

1. Open the `game.xml` layout file in the Eclipse layout resource editor and remove all existing controls from the layout.
2. Add a root `RelativeLayout` control and set its `background` attribute to `@drawable/background_paper`. Set its `layout_width` and `layout_height` attributes to `match_parent`. All subsequent controls will be added inside this control.

3. Within the `RelativeLayout` control, add an `ImageSwitcher` control with an id of `@+id/bookCoverImageSwitcher`. Set its `layout_width` and `layout_height` attributes to the dimension resource `wrap_content`. Set its `layout_centerHorizontal` and `layout_centerVertical` attributes to `true`. You can set the animations for switching between images directly via the XML using the `inAnimation` and `outAnimation` attributes. For example, you could set the `inAnimation` attribute to `@android:anim/fade_in` and the `outAnimation` attribute to `@android:anim/fade_out`, both of which are Android resources you don't need to create.
4. Add a `TextSwitcher` control with an id of `@+id/bookTitleTextSwitcher`. Set its `layout_width` and `layout_height` to `wrap_content`. Set its `layout_centerHorizontal` attribute to `true` and `layout_below` to `@+id/bookCoverImageSwitcher`. Again, set its `inAnimation` attribute to `@android:anim/fade_in` and the `outAnimation` attribute to `@android:anim/fade_out`.
5. Add a `TextView` control with an id of `@+id/haveYouReadText`. This will hold a prefix text, or somewhat of a header. Set its `layout_width` and `layout_height` attributes to `wrap_content`. Set its `layout_alignParentTop` and `layout_alignParentLeft` attributes to `true`. Set its `text` attribute to `@string/have_you_read`.
6. Add a `Button` control with an id of `@+id/yesButton`. Set `layout_width` and `layout_height` to `wrap_content`. Also, set its `layout_alignParentBottom` and `layout_alignParentRight` attributes to `true`. Set its `text` attribute to a resource string ("Yes") and tweak any other attributes to make the `Button` control look nice; specifically, you may want to set its `textSize` and `minWidth` to dimension resources created for this purpose. Finally, set its `onClick` to `onYesButton`; you will then need to create this method within your activity class to handle clicks.
7. Add a second `Button` control with an id of `@+id/noButton`. Set `layout_width` and `layout_height` to `wrap_content`. Also, set its `layout_alignParentBottom` and `layout_alignParentLeft` attributes to `true`. Set its `text` attribute to a resource string ("No") and tweak any other attributes to make the `Button` control look nice; specifically, you may want to set its `textSize` and `minWidth` to dimension resources created for this purpose. Finally, set its `onClick` to `onNoButton`; you will then need to create this method within your activity class to handle clicks.

At this point, save the `game.xml` layout file.

Working with ViewSwitcher Controls

For situations in which an activity is repeatedly going to be updating the content of a `View` control, the Android SDK provides a mechanism called a `ViewSwitcher` control. Using a `ViewSwitcher` is an efficient and visually interesting way to update content on a screen. A `ViewSwitcher` control has two children and handles transition from the currently visible child view to the next view to be displayed. The child `View` controls of a `ViewSwitcher` control are generated programmatically using `ViewFactory`.

There are two subclasses of the ViewSwitcher class:

- TextSwitcher—A ViewSwitcher control that allows swapping between two TextView controls
- ImageSwitcher—A ViewSwitcher control that allows swapping between two ImageView controls

Although a ViewSwitcher control only ever has two children, it can display any number of View controls in succession. ViewFactory generates the content of the next view, such as the ImageSwitcher and TextSwitcher controls for iterating through the book covers and titles.

Initializing Switcher Controls

Now let's turn our attention to the QuizGameActivity class and wire up the switchers. Begin by defining two private member variables within the activity class:

```
private TextSwitcher mBookTitleSwitcher;
private ImageSwitcher mBookCoverSwitcher;
```

You should initialize these switcher controls within the onCreate() method of the activity. To configure a switcher, use the setFactory() method and supply your custom ViewFactory class (android.widget.ViewSwitcher.ViewFactory). For example,

```
mBookTitleSwitcher = (TextSwitcher) findViewById(R.id.bookTitleTextSwitcher);
mBookTitleSwitcher.setFactory(new BookTitleSwitcherFactory());
```

```
mBookCoverSwitcher = (ImageSwitcher) findViewById(R.id.bookCoverImageSwitcher);
mBookCoverSwitcher.setFactory(new BookCoverSwitcherFactory());
```

Implementing Switcher Factory Classes

Now you need to create two classes: BookTitleSwitcherFactory and BookCoverSwitcherFactory. These can be inner classes within the activity itself and should implement the ViewSwitcher.ViewFactory class.

The ViewFactory class has one required method that you must implement: the makeView() method. This method must return a View of the appropriate type. For example, ViewFactory for TextSwitcher should return a properly configured TextView, whereas ViewFactory for ImageSwitcher would return ImageView. You could implement the makeView() method to build up and return the appropriate TextView or ImageView control programmatically, or you could create a simple layout resource as a template for your control and load it using a layout inflater. The second method makes for cleaner code, so that's what we'll do here.

Let's begin with the ImageSwitcher. First, create a layout resource called /res/layout/book_cover_switcher_view.xml, as follows:


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

Now, within the `QuizGameActivity` class, here's an implementation of a `ViewFactory` control for an `ImageSwitcher` control that you could use to generate each book cover on the game play screen:

```
private class BookCoverSwitcherFactory implements ViewSwitcher.ViewFactory {
    public View makeView() {
        ImageView imageView = (ImageView) LayoutInflater.from(
            getApplicationContext()).inflate(
                R.layout.book_cover_switcher_view,
                mBookCoverSwitcher, false);
        return imageView ;
    }
}
```

Note that the source data, or contents, of the view have not been configured in the `makeView()` method. Instead, consider this a template that the `ViewSwitcher` control will use to display each child view.

Similarly, you must create a layout resource for your `TextView`, such as `/res/layout/book_title_switcher_view.xml`, as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:textColor="@color/quiz_book_title_color"
    android:textSize="@dimen/game_book_title_text_size" >
</TextView>
```

Next, implement the `BookTitleSwitcherFactory` class (as an inner class within the activity) such that it loads the `TextView` from the layout resource you just created:

```
private class BookTitleSwitcherFactory implements ViewSwitcher.ViewFactory {
    public View makeView() {
        TextView textView = (TextView) LayoutInflater.from(
            getApplicationContext()).inflate(
                R.layout.book_title_switcher_view, mBookTitleSwitcher,
                false);
        return textView;
    }
}
```

Much like the `MyImageSwitcherFactory` implementation, the `MyTextSwitcherFactory` also implements the `makeView()` method—this time, generating the appropriate `TextView` control.

Updating the TextSwitcher Control

The `TextSwitcher` control enables an activity to animate between two `TextView` controls. So far, you have

- Included a `TextSwitcher` control in your layout resource file
- Added a member variable for the `TextSwitcher` to your activity class
- Initialized the `TextSwitcher` and implemented its factory class

You're almost done wiring up your `TextSwitcher` control. All you need to do now is determine under what conditions the `TextView` control must be updated. It makes sense to update the book data in two circumstances: when the game screen is first loaded and after the user clicks one of the answer buttons, moving on to the next book. In both circumstances, your activity should determine which book to display and then update the `TextSwitcher` object, which then animates out the previous `TextView` control (if applicable) and animates in a newly generated `TextView` control in its place.

Whether you are initializing the `TextSwitcher` with the first book title or the tenth book title, the method call is the same:

```
mBookTitleSwitcher.setCurrentText("First Book Title");
```

Calling the `setCurrentText()` method causes `BookTitleSwitcherFactory` to generate a new `TextView` control with the `String` parameter. Now, in the case of the `QuizGameActivity` class, you would not provide literal string data, but instead determine the next book from the XML data and supply it to the `TextSwitcher` control.

Updating the ImageSwitcher Control

The `ImageSwitcher` control is wired up in a very similar fashion to the `TextSwitcher` control. You have already defined, configured, and initialized the control. All that's left is to update the image content under the right circumstances. These circumstances are the same as the `TextSwitcher` control: when the game screen is first loaded or after a user clicks a `Button` control. Whether you are initializing the `ImageSwitcher` with the first book cover image, or the tenth book cover image, the method call is the same:

```
mBookCoverSwitcher.setImageDrawable(drawable);
```

Calling the `setImageDrawable()` method causes `MyImageSwitcherFactory` to generate a new `ImageView` control with the `Drawable` parameter. The `ImageSwitcher` class also has methods for loading images via resource identifiers and other methods. In this case, you want to create a `Drawable` object by loading an image from a remote URL. To do this, you will need to take some additional steps:

1. Retrieve the book cover image address from the XML books.
2. Generate a properly configured URL object.
3. Open and decode an input stream to the URL into a `Bitmap` object.
4. Create a `BitmapDrawable` from the original `Bitmap` for use as the `Drawable` parameter to the `setImageDrawable()` method.

These steps are easily consolidated into a helper function in the `QuizGameActivity` class:

```
private Drawable getBookCoverImageDrawable(int bookNumber) {
    Drawable image;
    URL imageUrl;
    try {
        imageUrl = new URL(getBookCoverUrl(bookNumber));
        InputStream stream = imageUrl.openStream();
        Bitmap bitmap = BitmapFactory.decodeStream(stream);
        image = new BitmapDrawable(getResources(), bitmap);
    } catch (Exception e) {
        Log.e(DEBUG_TAG, "Decoding Bitmap stream failed");
        image = getResources().getDrawable(R.drawable.nobook);
    }
    return image;
}
```

The `getBookCoverImageDrawable()` helper method takes a question number, retrieves the appropriate question image URL (using another helper method called `getBookCoverUrl()` method, whose full implementation is provided in the sample source code), generates a URL object from the String representation of the web address, opens and decodes the data stream into a `Bitmap` object, and finally generates a `BitmapDrawable` object for use by the `ImageSwitcher`. Finally, using the stream methods requires the `android.permission.INTERNET` permission, which needs to be added to your application Android manifest file in order for this method to function properly.

Wiring Up Game Logic

The *Have You Read That?* application has an open-ended set of books to ask about. Therefore, you cannot save all the books as resources, but instead need to develop a simple way to get new books on-the-fly. Also, by storing the complete (yet growing) set of books and their images in a remote location, you streamline the application on the handset, saving valuable disk space in exchange for using some bandwidth.

In the final version of the application, you will retrieve new batches of books from the Internet. For now, however, you can retrieve two batches of books from local XML files, simulating this effect without implementing the networking code required for the full solution. The application can keep a working set of books in memory, and new batches of books can be loaded as required.

To implement the game logic for the game screen, follow these steps:

1. Update `SharedPreferences` with game state settings.
2. Handle the retrieval and parsing batches of books (XML) into a relevant data structure, such as a `Hashtable`.
3. Implement `Button` click handling to drive the `ImageSwitcher` and `TextSwitcher` updates as well as the game logic.
4. Handle edge cases, such as when no more books are available.

The following subsections describe these steps in detail. The full implementation of the game logic, including loading and parsing the XML book data, loading them into an appropriate data structure, such as a `Hashtable` of `Book` objects (a helper class you will need to define), is too lengthy for complete coverage in this lesson, but we give you most of the basics in this section.

However, as the reader, you have everything you need to complete this task yourself. That said, if you get stuck or aren't looking for a challenge, just review and reproduce the implementation found in the `QuizGameActivity` class of the source code. For those looking for more of a challenge, feel free to make your own modifications or improvements to the game logic, but keep in mind that we will revisit this class in future lessons to add network support and other advanced features, and you will then have to support and modify this code.

Adding Game State Settings to the `SharedPreferences`

To keep track of game state, add two more `Integer` settings to the application `SharedPreferences`: the game score and the current book number. To add these preferences, first declare the preference name `String` values to the `QuizActivity.java` class:

```
public static final String GAME_PREFERENCES_SCORE = "Score"; // Long
public static final String GAME_PREFERENCES_CURRENT_BOOK = "CurBook"; // Int
```

Next, define the `SharedPreferences` object as a member variable of the `QuizGameActivity` class:

```
private SharedPreferences mGameSettings;
```

Initialize the `mGameSettings` member variable in the `onCreate()` method of the `QuizGameActivity` class:

```
mGameSettings = getSharedPreferences(GAME_PREFERENCES, Context.MODE_PRIVATE);
```

Now, you can use `SharedPreferences` throughout the class, as needed, to read and write game settings, such as the current book and the game score. For example, you could get the current book by using the `getInt()` method of `SharedPreferences`, as follows:

```
int startingBookNumber = mGameSettings.getInt(
    GAME_PREFERENCES_CURRENT_BOOK, 0);
```

If you attempt to get the current book and it has not yet been set, then you are at the beginning of the game and should start at the first book and update the current book number accordingly. Each time the user answers whether he or she read a book (and clicks a `Button` control), the current book number should be updated. If the Yes button was clicked, the score preference should also be incremented at the same time.

Retrieving, Parsing, and Storing Book Data

We could load up every book from the start, but this architecture is not terribly efficient, especially if the list is large or unbounded. Instead, we aim for a more flexible approach: When the *Have You Read That?* application runs out of books to display to the user, it attempts to retrieve a new batch of books. This architecture makes enabling networking for the application more straightforward in future chapters because the parsing of the XML remains the same, but the application requires less memory as it manages only a small, rolling “batch” of books at a given time. Each batch of books arrives as a simple XML file, which needs to be parsed.

Declaring Helpful String Literals for XML Parsing

Take a moment to review the XML format used by the book batches. To parse the book batches, you need to add several `String` literals to represent the XML tags and attributes to the `QuizActivity.java` class:

```
public static final String XML_TAG_BOOK_BLOCK = "books";
public static final String XML_TAG_BOOK = "book";
public static final String XML_TAG_BOOK_ATTRIBUTE_NUMBER = "number";
public static final String XML_TAG_BOOK_ATTRIBUTE_TITLE = "title";
public static final String XML_TAG_BOOK_ATTRIBUTE_COVERURL = "coverUrl";
```

While you are at it, define the default batch size to simplify allocation of storage for books while parsing the XML:

```
public static final int BOOK_BATCH_SIZE = 2;
```

The size of the book batch is flexible, but this works well for our small mock XML.

Storing the Current Batch of Books in a `Hashtable`

You can store the current batch of books in memory by using a simple but powerful data structure—in this case, we recommend the `Hashtable` class (`java.util.Hashtable`). A `hashtable` is simply a data structure with key-value pairs, handy for quick lookups. For game logic purposes, it makes sense for the key to be the book number and the value to be the book data (the book title and cover image URL). To store the book data, you need to create a simple data structure. Within the `QuizGameActivity` class, implement a simple helper class called `Book` to encapsulate a single piece of book data:

```
private class Book {
    int mNumber;
    String mTitle;
```

```
String mImageUrl;

public Book(int bookNum, String bookTitle, String bookCoverUrl) {
    mNumber = bookNum;
    mTitle = bookTitle;
    mImageUrl = bookCoverUrl;
}
}
```

Next, declare a `Hashtable` member variable within the `QuizGameActivity` class to hold a batch of `Book` objects in memory after you have parsed a batch of XML:

```
private Hashtable<Integer, Book> mBooks;
```

You can instantiate the `Hashtable` member variable in the `onCreate()` method of the `QuizGameActivity` class, as follows:

```
mBooks = new Hashtable<Integer, Book>(BOOK_BATCH_SIZE);
```

Now, whenever books are needed, retrieve the latest XML parcel, parse the XML, and stick the `Book` data into the `Hashtable` for use throughout the `QuizGameActivity` class. To save new key-value pairs to the `Hashtable` class, use the `put()` method. To retrieve a specific `Book` object by its book number, use the `get()` method. For example, to retrieve the current book information, check what the book number is (from the `SharedPreferences` setting you created earlier), and then make the following `get()` call:

```
Book curBook = (Book) mBooks.get(bookNumber);
```

You can check for the existence of a specific book in the `Hashtable` member variable by book number using the `containsKey()` method. This can help determine if it's time to retrieve a new batch of books and handle the case where no new books are available.

For a full implementation of retrieving and parsing the XML book data and storing it within the `Hashtable` data structure, consult the source code that accompanies this chapter. This implementation is available primarily within the `loadBookBatch()` helper method of the `QuizGameActivity` class and relies solely on common Java classes and the XML parsing method you learned in Chapter 9, "Developing the Help and Scores Screens."

Handling Button Clicks and Driving the Game Forward

The two `Button` controls on the game screen are used to drive the `ImageSwitcher` and `TextSwitcher` controls which, in turn, represent the book displayed to the user. Each time the user clicks a `Button` control, any score changes are logged, the current book number is incremented, and the `ViewSwitcher` controls are updated to display the next book. In this way, the `Button` controls drive the progress forward and the user progresses through the game.

Back when you were designing the `game.xml` layout, you set the `onClick` attributes of both `Button` controls; now it is time to implement these click handlers within the

QuizGameActivity class. There is little difference between the handling of the Yes and No Button controls:

```
public void onNoButton(View v) {
    handleAnswerAndShowNextBook(false);
}

public void onYesButton(View v) {
    handleAnswerAndShowNextBook(true);
}
```

Both Button controls rely upon a helper method `handleAnswerAndShowNextBook()`. This method is at the heart of the game logic; here, you must log the game state changes and handle all `ImageSwitcher` and `TextSwitcher` update logic. Here is pseudocode for this method:

```
private void handleAnswerAndShowNextBook(boolean bAnswer) {
    // Load game settings like score and current book
    // Update score if answer is "yes"
    // If answer is no, optionally show dialog offering a way to buy book on Amazon
    // Load the next book, handling if there are no more books
}
```

Now let's work through the pseudocode and implement this method. First, retrieve the current game settings, including the game score and the next book number, from `SharedPreferences`:

```
int curScore =
    mGameSettings.getInt(GAME_PREFERENCES_SCORE, 0);
int nextBookNumber =
    mGameSettings.getInt(GAME_PREFERENCES_CURRENT_BOOK, 1) + 1;
```

Next, save off the current book number to the `SharedPreferences`. If the user clicked the Yes button (and therefore the incoming parameter is `true`), update the score and save it. Then, commit these preference changes:

```
Editor editor = mGameSettings.edit();
editor.putInt(GAME_PREFERENCES_CURRENT_BOOK, nextBookNumber);

if (bAnswer == true) {
    editor.putInt(GAME_PREFERENCES_SCORE, curScore + 1);
}
editor.commit();
```

Now it's time to move on to the next book. First, check whether the next book is available in the `Hashtable` using the `containsKey()` method. If there are no remaining books in the hashtable, retrieve a new batch of books:

```
if (mBooks.containsKey(nextBookNumber) == false) {
    // Load next batch before checking again
}
```

```

try {
    loadBookBatch(nextBookNumber);
} catch (Exception e) {
    Log.e(DEBUG_TAG, "Loading updated book batch failed", e);
}
}

```

The `loadBookBatch()` helper method simply retrieves the next XML parcel, parses it, and shoves the new batch of `Book` data into the `Hashtable` for use by the application. See the source code that accompanies this chapter for the full implementation of this method if you require another example of XML parsing.

Returning to the topic at hand, you should now have a fully loaded `Hashtable` of book data and a new book to pose to the user. Update the `TextSwitcher` and the `ImageSwitcher` controls with the text and image for the next book. If there is no book to display, handle this case as well:

```

if (mBooks.containsKey(nextBookNumber) == true) {
    // Update book text
    mBookTitleSwitcher.setText(getBookTitle(nextBookNumber));

    // Update book image
    Drawable image = getBookCoverImageDrawable(nextBookNumber);
    mBookCoverSwitcher.setImageDrawable(image);
} else {
    handleNoBooks();
}

```

We discuss the `handleNoBooks()` helper method in a moment, and the `getBookTitle()` method simply does as its name suggests, looking up the book title in the `Hashtable` by book number. But for now, if you have implemented all the bits described thus far, you should be able to run the application, launch the game screen, and answer which books you've read. The game screen should look something like Figure 12.3.

Addressing Edge Cases

If there are no more books available, you must inform the user. This case is handled by the `handleNoBooks()` helper method. This method is straightforward. It does the following:

- Displays an informative text message to the user stating that there are no books available using the `TextSwitcher` control
- Displays a clear error graphic to the user, indicating that there are no books available using the `ImageSwitcher` control
- Disables both `Button` controls

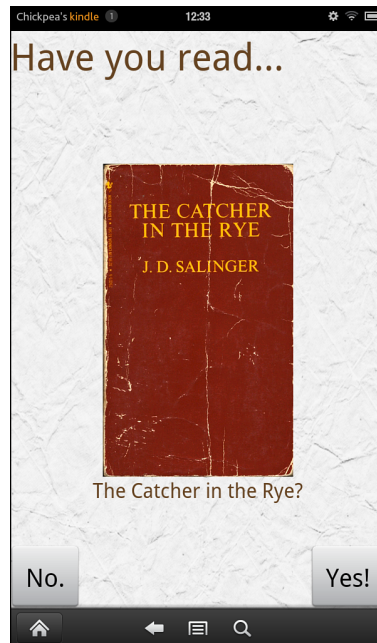


Figure 12.3 The Have You Read That? Game Screen

Here is the `handleNoBooks()` implementation:

```
private void handleNoBooks() {
    mBookTitleSwitcher.setText(getResources().getText(
        R.string.quiz_no_books));
    mBookCoverSwitcher.setImageResource(R.drawable.nobook);

    // Disable yes button
    Button yesButton = (Button) findViewById(R.id.yesButton);
    yesButton.setEnabled(false);

    // Disable no button
    Button noButton = (Button) findViewById(R.id.noButton);
    noButton.setEnabled(false);
}
```

When the application runs out of books, the game screen looks like in Figure 12.4. The user is informed that there are no more books available and he is not allowed to press any of the `Button` controls. Instead, the user must press the Back button and return to the main menu.

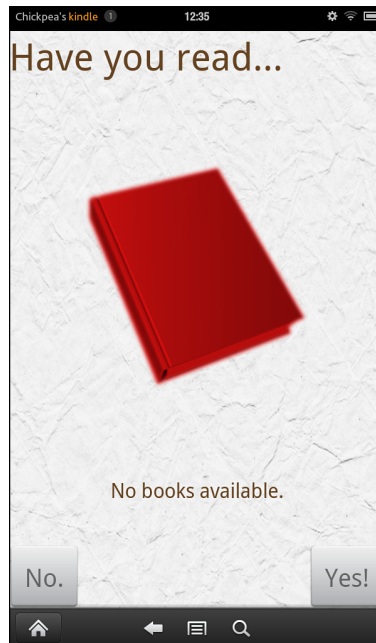


Figure 12.4 The Have You Read That? Game Screen When No Books Are Available

Summary

In this chapter, you implemented the most important screen of the Have You Read That? application: the game screen. You learned how to animate between view controls by using `ImageSwitcher` and `TextSwitcher`. You also got your first look at the various data structures available in the Android SDK and used a `Hashtable` member variable to store a batch of books parsed from XML. Finally, you used the application's `SharedPreferences` to keep track of settings and game state information.

Exercises

1. Review the sample code that accompanies this chapter, especially the `QuizGameActivity` class. Make sure that you understand each and every feature implemented in that class. If you are following along and building your own version of the application, ensure that all features are implemented in your own version of the activity.

2. Update the game screen to display the user's current score. This change requires the addition of a `TextView` control to the `game.xml` layout resource, which will need to be updated for each book displayed. Recall that the score is the number of Yes answers and is saved as a preference.
3. Modify the application to use a different data structure, like a `Map`, `Vector`, or linked list instead of a `Hashtable`.
4. [Advanced] Add a new option to the options menu of the game screen, created back in Chapter 8, "Implementing the Main Menu Screen," to reset the game. Make sure that you clear only the appropriate game settings upon a reset, not all application preferences, in `SharedPreferences`.

Adding Network Support

In this chapter, you enable the Have You Read That? application to handle two-way network communication. First, you concentrate your attention on downloading data from the Internet. Specifically, you learn the theory and design principles for networked applications. You learn about threading and progress bars. You modify the Have You Read That? application to retrieve batches of books and live score data from a remote application server. You then enhance the Have You Read That? application to upload player data, such as settings, scores, and avatars to the application server. You learn how to upload data to a network server, as well as a new way to offload important processing to a simple service that executes in the background.

Designing Network Applications

Although mobile devices have come a long way in terms of computing speed and data storage, servers can still provide valuable processing power for backing up data or for providing ease of data portability between devices or access to portions of larger datasets that can't be retained on the local device. By design, today's mobile devices can easily connect to networks and the Internet, and many rely on cloud-based services.

Most Android devices can connect to the Internet in multiple ways, including through 3G, 4G, and beyond networks or Wi-Fi connections. Android applications can use many of the most popular Internet protocols, including HTTP, HTTPS, TCP/IP, and raw sockets. The Android SDK gives developers access to these communication protocols, provided that the application has the appropriate permissions.

Working with an Application Server

Network-enabled applications often rely on an application server. The application server provides centralized data storage (a database) and high-performance processing power. Using a centralized server also allows the developer to implement a single server side with access from multiple client applications. For example, you could easily write iPhone, BlackBerry, and web versions of the Have You Read That? application that use the same backend application server. Score and friend data could then easily be shared across multiple client environments.

We developed a simple application server for use with this book, and the Have You Read That? application, for your use. There are many ways to develop an application server, the details of which are far beyond the scope of this book. However, we want to provide you with a comprehensive networking example, beyond that of connecting to some third-party web service, like a weather feed or something to that effect. Instead, we provide you with a fully functional open source server implementation, which you can choose to explore or not. The choice is yours; after all, you are reading this book to learn Android, so you may want to stay focused on the business at hand. However, others cannot help but want to know every detail. We tried to provide for both types of learners.

Here are the basics. Our network server implementation leverages a simple, scalable server powered by Google App Engine (<http://code.google.com/appengine/>) with Java and servlets. Although the implementation details of the application server are beyond the scope of this book, it can help to understand how it was designed. We provide the source code in open source form at <http://goo.gl/F9Fx5>.

Think of this application server as a black box with the following attributes:

- The application server is always on and available for users. It's not in our broom closet.
- The application server is ours. We don't have to worry about a third party changing its services without warning and breaking the code. Because we control it, we can keep it stable.
- The application server is remotely accessed via HTTP, which is perhaps the most common communication protocol used by Internet-connected devices these days.
- The application server stores data, such as player settings, scores, and books.
- The application server can be queried for information, such as top scores or batches of books.
- The application server uses JavaServer Pages (JSP) to handle HTTP requests and return the appropriate results in the XML format that the Have You Read That? application expects—that is, the same XML schema that the mock data uses.

You could create an application server that has different characteristics than the one described here. For example, you could create a SQL database-driven application server using MySQL and PHP instead, or you could use a number of other technologies. You could run it out of your broom closet, if you were so inclined. However, creating a network server from scratch isn't something we expect readers to do. (Although, depending on your background, you might be more than proficient at this.) Therefore, feel free to use the network server we provided for the network portions of the sample application; in truth, we recommend doing so. Again, we are not expecting you to develop your own network server for use with this book.

Managing Lengthy Network Operations

Connecting, downloading, parsing, displaying content... Network operations can take time. If an operation takes too long, the Android operating system may shut down the offending

application for lack of response (the dreaded “force close”). Therefore, all network-related calls should be handled asynchronously, separately from the main UI thread. This can be accomplished by using the Java `Thread` class or by using the helpful Android `AsyncTask` class, which we discuss later in this chapter. In fact, in versions of Android newer than that found on the Kindle Fire platform, it’s a requirement that all network operations be performed off the main thread, no matter how small they are. It’s a good practice to get in the habit of doing this.

Informing the User of Network Activity

In this chapter, we focus on the simple task of querying the application server and retrieving the XML returned from the query. Networking support does not necessitate any specific user interface or layout updates to the Have You Read That? application. However, any time the user is required to wait for an operation that may take time—for example, for XML data to be downloaded from a remote server and parsed—it is important to inform the user that something is happening. Developers can use a visual mechanism, such as an indeterminate `ProgressBar` control, for this purpose. Otherwise, the user may sit there wondering what’s going on and even abandon the application, thinking it’s hung or crashed.

Developing Network Applications

Developers who enable network support in their applications need to be aware of a number of issues and follow a number of guidelines. These issues are similar to those faced when enabling Location-Based Services (LBS) features in an application. User privacy concerns, device performance degradation, and unwanted network data charges are all common issues to consider when developing network applications. Also, network connectivity (availability, strength, and quality) is not guaranteed, so enabling network support gives your application a variety of opportunities to fail, so you want to design your application to fail gracefully.

Android devices address some of these issues, in part through permissions, but much of the burden of managing the impact of network features and performance falls on the developer. Here are some guidelines for developers leveraging network features within applications:

- Use network services only when they are needed and cache data locally whenever possible.
- Inform the user when collecting and using sensitive data, as appropriate.
- Allow the user to configure and disable features that might adversely affect his or her experience when using your application. For example, develop an airplane mode for your application to allow the user to enjoy your application without accessing a remote server.
- Gracefully handle events such as no network coverage. Your application is more valuable to the user if it is useful even without an Internet connection.
- Review your application’s network functionality once your application is stable, and make performance improvements.

- Consider including a privacy message as part of your application's terms of use. Use this opportunity to inform the user about what data is collected from the user, how it will and will not be used, and where it is stored (for example, on a remote application server).

Enabling Network Testing on the Emulator

You do not need to make any changes to the emulator to write network-enabled applications. The emulator will use the Internet connection provided by your development computer and simulate true network coverage. Also, the emulator has a number of settings for simulating network latency and network transfer speeds, which is useful for some devices, but if your development machine is connected via the same Wi-Fi as your Kindle Fire, network performance may be similar. For details on the network debugging features of the emulator, see the Android emulator documentation.

Testing Network Applications on Kindle Fire

As usual, the best way to test network-enabled applications is on the target Android device. There are a number of network-related settings on most Android devices; current Kindle Fire devices have just Wi-Fi. You configure these settings by tapping the Settings icon and choosing Wi-Fi. From there, you can turn off Wi-Fi, which is a useful test for most network applications.

Accessing Network Services

The Android platform has a wide variety of networking libraries. Those accustomed to Java networking will find the `java.net` package familiar. There are also some helpful Android utility classes for various types of network operations and protocols. Developers can secure network communication by using common technologies, such as SSL and HTTPS.

Planning Have You Read That? Network Support

So far, you have supplied only mock XML data in the Have You Read That? application. Now it's time to modify the application to contact a remote application server to get live data. To do this, you need to learn about the networking features available on the Android platform, as well as how to offload tasks from the main UI thread and asynchronously execute them.

Two classes of the Have You Read That? application need to download information from an application server:

- `QuizScoresActivity`—This class needs to download score information.
- `QuizGameActivity`—This class needs to download each batch of books.

To enable the Have You Read That? application to handle live data, you need to access an application server and add networking functionality to the client Android application. The fully

functional sample code for this lesson is provided on this book's websites. Feel free to follow along.

Setting Network Permissions

To access network services on an Android device, you must have the appropriate permissions. An Android application can use most networking services only if it is granted the appropriate `<uses-permission>` settings configured in the Android manifest file. The following are the three most common permission values used by applications leveraging the network:

- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.CHANGE_NETWORK_STATE`

There are a number of other permissions related to networking, including those that allow access and changes to Wi-Fi state and network state. Some applications may also use the `android.permission.WAKE_LOCK` permission to keep the device from sleeping—useful for operations that should be handled without interruption from the device sleeping while the user is watching them finish.

The Have You Read That? application does not require many network permissions in order to complete its networking tasks. The `android.permission.INTERNET` permission will suffice.

Checking Network Status

The Android SDK provides utilities for gathering information about the current state of a network. This is useful for determining whether a network connection is available before trying to use a network resource. By validating network connectivity before attempting to make a connection, you can avoid many of the failure cases common in mobile device networking applications and provide your end users with a more pleasant user experience.

Using HTTP Networking

The most common network transfer protocol is Hypertext Transfer Protocol (HTTP). Most commonly used HTTP ports are open and available for use on Android device networks.

A fast way to get to a network resource is by retrieving a stream object to the content. Many Android data interfaces accept stream objects. One such example that you should now be somewhat familiar with is `XmlPullParser`. The `setInput()` method of the `XmlPullParser` class takes an `InputStream` object. Previously, you retrieved this stream from the XML resources. Now, however, you can get it from a network resource by using the simple URL class, as shown here:


```
URL xmlUrl = new URL("http://...xmlSourcepath...");
XmlPullParser bookBatch =
    XmlPullParserFactory.newInstance().newPullParser();
bookBatch.setInput(xmlUrl.openStream(), null);
```

That's it. The only magic here is determining what URL to use; in this case, the network server has a specially formatted URL for book batches (<http://hyreadthat.appspot.com/questions.jsp>) and another for score data (<http://hyreadthat.appspot.com/scores.jsp>).

Once the appropriate URL is formatted and the `XmlPullParser` is created, the parsing of the XML remains unchanged, because the format is no different and the `XmlResourceParser` used previously implements the `XmlPullParser` interface. After you have the question batches and score data downloading from the remote server, you can remove the mock XML resources from the project and the code that retrieves the XML resources.

Indicating Network Activity with Progress Bars

Network-enabled applications often perform lengthy tasks, such as connecting to remote servers and downloading and parsing data. These tasks take time, and the user should be aware that these activities are taking place. As discussed earlier in this lesson, a great way to indicate that an application is busy doing something is to show some sort of progress indicator. The Android SDK provides two basic styles of the `ProgressBar` control to handle determinate and indeterminate progress.

Displaying Indeterminate Progress

The simplest `ProgressBar` control style is a circular indicator that animates. This kind of progress bar does not show progress per se, but informs the user that something is happening. Use this style of progress bar when the length of the background processing time is indeterminate.

Displaying Determinate Progress

When you need to inform the user of specific milestones in progress, use the determinate progress bar control. This control displays as a horizontal progress bar that can be updated to show incremental progress toward completion. To use this progress indicator, use the `setProgress()` method of the `ProgressBar` control.

As described later in this chapter, you can put progress bars in the application's title bar. This can save valuable screen space. You will often see this technique used on screens that display web content.

Displaying Progress Dialogs

To indicate progress in a dialog window, as opposed to adding a `ProgressBar` control to the layout of an existing screen, use the special `Dialog` class called `ProgressDialog`. For example,

use `ProgressDialog` windows (see Figure 13.1) in the *Have You Read That?* application to inform the user that data is being downloaded and parsed before displaying the appropriate screen of the application.

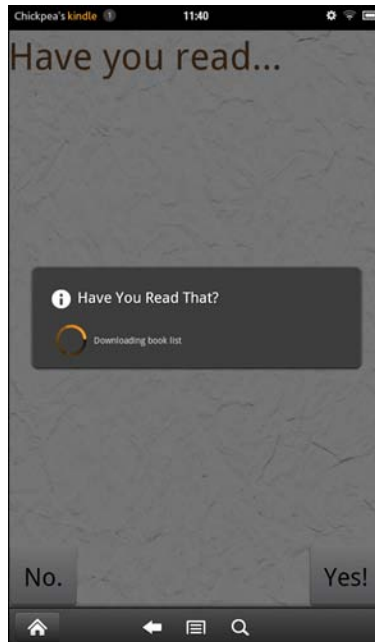


Figure 13.1 Informing the User that a New List of Books Is Being Downloaded

Here is the code needed to programmatically create and display the `ProgressDialog` class:

```
ProgressDialog mPleaseWaitDialog = ProgressDialog.show(QuizGameActivity.this,
    "Have You Read That?", "Downloading book list", true, true);
mPleaseWaitDialog.setOnCancelListener(new OnCancelListener() {
    public void onCancel(DialogInterface dialog) {
        BookListDownloaderTask.this.cancel(true);
    }
});
```

Use the `dismiss()` method to dismiss the `pleaseWaitDialog` control when the background processing has completed:

```
mPleaseWaitDialog.dismiss();
```

Now you know how to create progress bars and display them in dialog windows using the `ProgressDialog` control. Because the indicated progress will actually be taking place asynchronously, it's time to turn our attention to background processing.

Running Tasks Asynchronously

Despite rapidly evolving technology, mobile wireless networks still provide relatively slow Internet connections compared to those found in personal computers. Your Android applications must be responsive, so you should always move all network operations off the main UI thread and onto a secondary “worker” thread. The Android platform provides two easy methods for achieving this:

- **AsyncTask**—This abstract class can be used to offload background operations from the UI thread easily. Operations are managed by this helper class, making it the most straightforward for beginners and those not familiar with Java threading.
- **Thread and Handler**—These classes can be used together to handle concurrent processing and communicating with the UI thread’s message queue. This advanced method allows more flexibility in terms of implementation, but you, as the developer, are responsible for managing thread operations appropriately.

For the Have You Read That? application, the `AsyncTask` class is most appropriate because it’s the most straightforward to implement.

Using AsyncTask

The Android SDK includes the `AsyncTask` class (`android.os.AsyncTask`) to help manage background operations that will eventually post back to the UI thread.

Instead of using handlers and creating threads, you can simply create a subclass of the `AsyncTask` class and implement the appropriate callback methods:

- `onPreExecute()`—This method runs on the UI thread before background processing begins.
- `doInBackground()`—This method runs in the background and is where all the real work is done.
- `publishProgress()`—This method, called from the `doInBackground()` method, periodically informs the UI thread about the background process progress. This method sends information to the UI process. Use this opportunity to send updated progress for a progress bar that the user can see.
- `onProgressUpdate()`—This method runs on the UI thread whenever the `doInBackground()` method calls `publishProgress()`. This method receives information from the background process. Use this opportunity to update a `ProgressBar` control that the user can see.
- `onPostExecute()`—This method runs on the UI thread once the background processing is completed.

When launched with the `execute()` method, the `AsyncTask` class handles processing in a background thread without blocking the UI thread.

Using Threads and Handlers

If you need to control a thread yourself, use the `Thread` class (`java.lang.Thread`) in conjunction with a `Handler` object (`android.os.Handler`). The `Activity` class that owns the thread is responsible for managing the lifecycle of the thread. Generally speaking, the `Activity` includes a member variable of type `Handler`. Then, when the thread is instantiated and started, the `post()` method of the `Handler` is used to communicate with the main UI thread.

Downloading and Displaying Score Data

Let's begin by creating an asynchronous task for downloading the score datasets on the Have You Read That? scores screen. Because the top scores and friends' scores data is very similar, there's no reason not to create just one kind of `AsyncTask` class to handle both types of downloads. You can then create two instances of the task: one for top scores and another for friends' scores. This process involves extending the `AsyncTask` class, implementing the appropriate callbacks, creating two instances of the task (top scores and friends' scores), and then starting those tasks.

The network server has a JSP page for handling score requests. Define the appropriate URL strings in your `QuizActivity` class for use in the appropriate activities. For example,

```
public static final String HYRT_SERVER_BASE = "http://hyreadthat.appspot.com/";
public static final String HYRT_SERVER_SCORES_XML_PATH = HYRT_SERVER_BASE+ "scores.
➤jsp";
```

This JSP page can take one parameter when necessary: the user's player identifier. This identifier, which will be stored on the network server in the next lesson, helps the network server filter to the player's friends' scores. For this lesson, you'll want to supply a player identifier of 1. In the next chapter, you'll flesh out this feature and be able to query for your own data. So, for example, to get the top scores, use the following URL for your query:

```
http://hyreadthat.appspot.com/scores.jsp
```

Whereas if you wanted the user's friends' scores, you would tack on the `?playerId=` query variable with the player identifier, like this:

```
http://hyreadthat.appspot.com/scores.jsp?playerId=2008
```

Extending `AsyncTask` for Score Downloads

Now, let's work through the steps required to create an `AsyncTask` within the `QuizScoresActivity` class to handle the downloading and parsing of XML score information. Begin by creating an inner class within `QuizScoresActivity` called `ScoreDownloaderTask`, which extends the `AsyncTask` class within the `QuizScoresActivity` class:

```
private class ScoreDownloaderTask extends AsyncTask<Object, String, Boolean> {
    private static final String DEBUG_TAG = "ScoreDownloaderTask";
    TableLayout table;
    // TODO: Implement AsyncTask callback methods
}
```

Because you will be populating a `TableLayout` control as part of this background task, it makes sense to add a handy member variable within `ScoreDownloaderTask`. While you're at it, override the `DEBUG_TAG` string so that events logged within the asynchronous task will have a unique tag in `LogCat`.

At this time, you can also move the XML parsing helper methods from the `QuizScoresActivity` class, such as the `processScores()` and `insertScoreRow()`, to the `ScoreDownloaderTask` class. This allows all XML parsing and processing to occur within the asynchronous task instead of on the main thread. You should leave the `initializeHeaderRow()` and `addTextToRowWithValues()` methods in the `QuizScoresActivity` class, because they may be used even if the scores cannot be downloaded.

Finally, create two member variables within the `QuizScoresActivity` class to represent the two score sets to download:

```
private ScoreDownloaderTask allScoresDownloader;
private ScoreDownloaderTask friendScoresDownloader;
```

Starting the Progress Indicator with `onPreExecute()`

Next, you need to implement the `onPreExecute()` callback method of your `ScoreDownloaderTask` class, which runs on the UI thread before background processing begins. This is the perfect place to demonstrate adding an indeterminate progress indicator to the title bar:

```
@Override
protected void onPreExecute() {
    mProgressCounter++;
    QuizScoresActivity.this.setProgressBarIndeterminateVisibility(true);
}
```

There are two tabs of scores. Each tab's scores are downloaded separately, and you want the progress indicator to display until both are complete. Thus, you should create a counter member variable at the `QuizScoresActivity` class level (not the `ScoreDownloaderTask` level), called `mProgressCounter`, to track each download. In this way, you could add any number of tabs, and the indicator would still show and disappear at the correct time.

To use an indeterminate progress bar on the title bar of your screen, you need to add the following code to the `onCreate()` method of the `QuizScoresActivity`:

```
requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
```

You must call this method before you call the `setContentView()` method.

Clearing the Progress Indicator with `onPostExecute()`

Next, implement the `onPostExecute()` callback method, which runs on the UI thread after background processing completes. Specifically, when the application has completed all parsing and displaying, it can hide the progress indicator shown in the title bar if all the tasks are complete, as determined by the `mProgressCounter` variable:

```
@Override
protected void onPostExecute(Boolean result) {
    Log.i(DEBUG_TAG, "onPostExecute");
    mProgressCounter--;
    if (mProgressCounter <= 0) {
        mProgressCounter = 0;
        QuizScoresActivity.this.
            setProgressBarIndeterminateVisibility(false);
    }
}
```

Again, all this callback does is decrement the counter and terminate the progress indicator, if necessary.

Handling Cancellation with `onCancelled()`

You can handle cancellation of the background processing by overriding the `onCancelled()` callback method. The `onCancelled()` method runs on the UI thread and, if it's called, it means that the `onPostExecute()` method will not be called. Thus, any cleanup must be performed here. For this example, we perform the following operation:

```
@Override
protected void onCancelled() {
    Log.i(DEBUG_TAG, "onCancelled");
    mProgressCounter--;
    if (mProgressCounter <= 0) {
        mProgressCounter = 0;
        QuizScoresActivity.this.
            setProgressBarIndeterminateVisibility(false);
    }
}
```

The `onCancelled()` method is called when the `cancel()` method of `AsyncTask` is called. This does not happen automatically. Instead, good practice is for the `Activity` that owns the asynchronous task to cancel tasks when they are no longer needed. For the scores screen, you'll want to cancel the tasks if they're still running when the user leaves the screen for any reason. That is, you cancel them in the `onPause()` callback method of the `QuizScoresActivity` class, as shown here:

```
@Override
protected void onPause() {
    if (allScoresDownloader != null &&
```

```

        allScoresDownloader.getStatus() !=
        AsyncTask.Status.FINISHED) {
            allScoresDownloader.cancel(true);
        }
        if (friendScoresDownloader != null &&
            friendScoresDownloader.getStatus() !=
            AsyncTask.Status.FINISHED) {
            friendScoresDownloader.cancel(true);
        }
        super.onPause();
    }
}

```

Handling Processing with `doInBackground()`

Now, it is time to identify what processing should run asynchronously. For this example, it is the downloading and parsing of some XML from a network server. Override the `doInBackground()` callback method, which is where all the background processing takes place. Any methods called within `doInBackground()` will not block the main UI thread. Here's a sample implementation of the `doInBackground()` method, with exception handling removed for clarity:

```

@Override
protected Boolean doInBackground(Object... params) {
    boolean result = false;
    String pathToScores = (String) params[0];
    table = (TableLayout) params[1];
    XmlPullParser scores = null;
    URL xmlUrl = new URL(pathToScores);
    scores = XmlPullParserFactory.newInstance().newPullParser();
    scores.setInput(xmlUrl.openStream(), null);
    if (scores != null) {
        processScores(scores);
    }
    return result;
}

```

Here, we use the flexible incoming parameters to supply the appropriate URL to the scores we want to download (top or friends). This string is used to generate the appropriate URL to the application server, and then an `XmlPullParser` instance is used to download the score data and parse it, as previously discussed.

You now need to make one subtle change to the `processScores()` helper method, which simply takes the `XmlPullParser` and parses the XML, to publish scores as they are parsed within the task, using the `publishProgress()` method:

```

private void processScores(XmlPullParser scores)
    throws XmlPullParserException, IOException {
    int eventType = -1;
    boolean bFoundScores = false;

    // Find Score records from XML
    while (eventType != XmlResourceParser.END_DOCUMENT) {
        if (eventType == XmlResourceParser.START_TAG) {

            // Get the name of the tag (eg scores or score)
            String strName = scores.getName();

            if (strName.equals("score")) {
                bFoundScores = true;
                String scoreValue =
                    scores.getAttributeValue(null, "score");
                String scoreRank =
                    scores.getAttributeValue(null, "rank");
                String scoreUserName =
                    scores.getAttributeValue(null, "username");
                publishProgress(scoreValue, scoreRank, scoreUserName);
            }
        }
        eventType = scores.next();
    }

    // Handle no scores available
    if (bFoundScores == false) {
        publishProgress();
    }
}

```

The `publishProgress()` method can be called anytime within the `doInBackground()` method to cause the `onProgressUpdate()` callback method to be called. This allows the background process to communicate with the UI thread, which can publish updates to the screen, whereas the background task cannot directly act upon the screen. Now let's implement the `onProgressUpdate()` callback method.

Handling Progress Updates with `onProgressUpdate()`

You can update the UI thread with background progress information by overriding the `onProgressUpdate()` callback method of the `ScoreDownloaderTask` class. This method enables you to display score data as it is parsed, instead of parsing all score data and then displaying it all in one go when the asynchronous task is completed. Users appreciate this because, as you may have noticed, users are impatient.

Update the `onProgressUpdate()` method at this time. Pass in the new score just parsed using the flexible method parameters and insert a new row in the score `TableLayout` control, like this:

```
@Override
protected void onProgressUpdate(String... values) {
    if (values.length == 3) {
        String scoreValue = values[0];
        String scoreRank = values[1];
        String scoreUserName = values[2];
        insertScoreRow(table, scoreValue, scoreRank, scoreUserName);
    } else {
        final TableRow newRow =
            new TableRow(QuizScoresActivity.this);
        TextView noResults =
            new TextView(QuizScoresActivity.this);
        noResults.setText(
            getResources().getString(R.string.no_scores));
        newRow.addView(noResults);
        table.addView(newRow);
    }
}
```

The `insertScoreRow()` method simply creates a new `TableRow` control and adds it to the `TableLayout` control. The array of values must be passed in the same order each time. This is because of how the `AsyncTask` Java template works.

Starting the `ScoreDownloaderTask`

The `ScoreDownloaderTask` class is now complete. Now, you just need to launch it. Do this by updating the `onCreate()` method of the `QuizScoresActivity` class to call the `ScoreDownloaderTask` class's `execute()` method when the screen first loads. The `execute()` method takes two parameters: the server web address and the table to populate with scores (a `TableLayout` control, as defined in the tabs):

```
public static final String HYRT_SERVER_BASE =
    "http://hyreadthat.appspot.com/";
public static final String HYRT_SERVER_SCORES_XML_PATH =
    HYRT_SERVER_BASE + "scores.jsp";
// ...
allScoresDownloader =
    new ScoreDownloaderTask();
allScoresDownloader.execute(HYRT_SERVER_SCORES_XML_PATH, allScoresTable);

Integer playerId = 2008;
```

```

if (playerId != -1) {
    friendScoresDownloader = new ScoreDownloaderTask();
    friendScoresDownloader.execute(
        HYRT_SERVER_SCORES_XML_PATH + "?playerId="
        + playerId, friendScoresTable);
}

```

Here, we are doing a couple of things: We define the network server URL to use for downloading scores. We instantiate the two `ScoreDownloaderTask` instances, one for top scores and the other for friends' scores. We then call the `execute()` method for each task. Don't worry too much about the `playerId` value yet. We discuss that in the next chapter, when you begin saving player data to the server. The player identifier is needed so that the appropriate friends' scores are downloaded. For now, feel free to use a player identifier with a value of 2008 to guarantee downloads from one of the test accounts on the server.

Downloading and Parsing Batches of Books

Now that you understand how to download data asynchronously, you can use the `AsyncTask` again within the `QuizGameActivity` to handle downloading and displaying the book batches on the game screen. This process is similar to the process involved in downloading score data. However, you will not publish progress as you go; instead, you will simply display a progress bar until all books in a given batch are downloaded.

The network server has a JSP page for handling question batch requests. Define the appropriate URL strings in your `QuizActivity` class for use in the appropriate activities. For example,

```

public static final String HYRT_SERVER_BASE =
    "http://hyreadthat.appspot.com/";
public static final String HYRT_SERVER_BOOKS_XML_PATH =
    HYRT_SERVER_BASE+ "questions.jsp";

```

This JSP page can take two parameters: The `max` parameter specifies the book batch size, and the `start` parameter specifies the starting book number to retrieve. Using these two values, you can request the next batch of books, depending on where the user is in the quiz. The query parameters must be specified in this order. So, for example, to query for 15 questions starting at question number 16, you would use the following URL for your query:

```
http://hyreadthat.appspot.com/questions.jsp?max=15&start=16
```

Extending `AsyncTask` for Book Downloads

Begin by creating an inner class called `BookListDownloaderTask` within the `QuizGameActivity` class that extends the `AsyncTask` class, like this:

```

private class BookListDownloaderTask
    extends AsyncTask<Object, String, Boolean> {
    private static final String DEBUG_TAG =

```

```

        "QuizGameActivity$BookListDownloaderTask";
    private int startingNumber;
    private ProgressDialog mPleaseWaitDialog;
    // TODO: Implement AsyncTask callback methods
}

```

The `BookListDownloaderTask` class requires several member variables, including its own custom debug tag, the starting book number, and a `ProgressDialog` to display background processing progress to the user when necessary.

Starting the Progress Dialog with `onPreExecute()`

Now you need to implement the `onPreExecute()` callback method. This is the perfect place to display a progress dialog that tells the user that the trivia questions are being downloaded. The user won't be able to do anything until the books are downloaded. Put a progress dialog over the game screen:

```

@Override
protected void onPreExecute() {
    mPleaseWaitDialog = ProgressDialog.show(QuizGameActivity.this,
        "Have You Read That?", "Downloading book list", true, true);
    mPleaseWaitDialog.setOnCancelListener(new OnCancelListener() {
        public void onCancel(DialogInterface dialog) {
            Log.d(DEBUG_TAG, "onCancel -- dialog");
            BookListDownloaderTask.this.cancel(true);
        }
    });
}

```

Although we used hard-coded strings here for clarity, a well-written application would use string resources for easy localization. A cancel listener is configured for the dialog. This allows the user to press the back button to cancel the dialog. When this happens, the `cancel()` method of the `AsyncTask` is called. This means that cancelling the dialog will now cancel the task, which will cancel the network activity.

Dismissing the Progress Dialog with `onPostExecute()`

Next, implement the `onPostExecute()` method. Now that the background processing has taken place, drop in the code you originally used to display the screen. This is also the perfect place to dismiss the progress dialog:

```

@Override
protected void onPostExecute(Boolean result) {

    Log.d(DEBUG_TAG, "Download task complete.");
    if (result) {
        displayCurrentBook(startingNumber);
    }
}

```

```

    } else {
        handleNoBooks();
    }

    mPleaseWaitDialog.dismiss();
}

```

You also need to handle the cancel operation by implementing the `onCancelled()` callback method of the `BookListDownloaderTask` class as well as the `onPause()` callback of the `QuizGameActivity` class, much like you did for the scores implementation.

Handling the Background Processing

Now you need to identify what processing should run asynchronously. Again, this is the downloading and parsing code. The following code (with exception handling removed for clarity) shows how to override the `doInBackground()` callback method:

```

@Override
protected Boolean doInBackground(String... params) {
    boolean result = false;
    startingNumber = (Integer)params[1];
    String pathToBooks = params[0] +
        "?max=" + BOOK_BATCH_SIZE + "&start=" + startingNumber;
    result = loadBookBatch (startingNumber, pathToBooks);
    return result;
}

```

Here, the background processing simply involves determining the appropriate book batch to download and calling the helper method `loadBookBatch()`. We use the flexible parameters of the `doInBackground()` callback method to pass in the `max` and `start` criteria. You'll want to move the `loadBookBatch()` method from `QuizGameActivity` into the `BookListDownloaderTask` class and modify it to contact the application server at the appropriate URL. Again, this is simply a matter of generating the appropriate URL parameters, opening the stream to the remote application server, and using the `XmlPullParser` to process the XML data as before. The parsing details remain unchanged. Unlike the scores implementation, there is no need to post progress for this task.

Starting the BookListDownloaderTask

Once you implement the `BookListDownloaderTask` class, you can update the `onCreate()` method of the `QuizGameActivity` class to call the `execute()` method of the `BookListDownloaderTask` class when the screen first loads. In this case, the `execute()` method takes two parameters: the server web address for book downloads and the starting book number (an `Integer`) for the batch to download:

```

BookListDownloaderTask downloader = new BookListDownloaderTask();
downloader.execute(HYRT_SERVER_BOOKS_XML_PATH, startingBookNumber)

```

For the full implementation of the book download task, including some code rearranging and cleanup, see the sample code provided on this book's websites.

Determining What Data to Send to the Server

So far, you have only downloaded data from the network server within the Have You Read That? application. Now it's time to upload player information to the application server, creating a new account if necessary. To do this, you need to learn how to use the Apache HTTP client features available on the Android platform, as well as how to add extra Apache libraries to the project—libraries that aren't available with the Android SDK.

Three features of the Have You Read That? application require uploading data to the application server or use related player data to retrieve the appropriate results:

- **QuizSettingsActivity**—This class needs to create a player record on the application server and upload player settings information, including the nickname and email address information. This will be a one-way upload. This is primarily because this is just a sample application. Some data is later used during the score retrieval.
- **QuizGameActivity**—This class needs to upload the player's score. The player's score information will be compiled with other players' data to compute the top scores and friends' scores data.
- **QuizScoresActivity**—This class needs to be updated to use a valid player identifier to retrieve friends' scores (even though we don't implement the friends feature until the next chapter).

The complete implementation of the sample code for this chapter is provided on this book's websites. You may want to follow along.

Keeping Player Data in Sync

The Have You Read That? application must be kept simple. Some of the player settings will be uploaded to the application server, while others are only important to the application client. The application server needs to be able to track usage and installations. Therefore, the application must generate a unique identifier, store it, and send it with each request. The first time the server sees a new player, it will generate a player identifier and send it down to the client. The player's identifier is the key upon which many features hinge: the ability to update the appropriate record on the server, the ability to retrieve the appropriate player's friends' scores, and the ability to tie a player name to a specific score.

There are many ways to create unique identifiers. One way is to use the `randomUUID()` method of the `UUID` class (`java.util.UUID`), like so:

```
String uniqueId = UUID.randomUUID().toString();
```

Using a random UUID is preferable to some other methods of getting a unique identifier, such as determining the device identifier through the `TelephonyManager`, because it will return a valid identifier, regardless of whether the device is a smartphone, tablet, or Android-powered toaster. This is especially important for devices like the Kindle Fire, which does not have telephony hardware. It also works on all versions of the Android SDK, unlike using the `Settings.Secure.ANDROID_ID` value that wasn't introduced until Android 2.2.

The next question is how to keep two copies of player data—the application shared preferences copy and the network server copy—in sync. For our simple application, we can keep the data “synchronized” by designating the application's version of the data as the “main source,” and the network server's version as a copy that won't be read from to write local data. In other words, we synchronize player data one way only: from the application to the network server, but not in reverse. This helps avoid some of the design complexity of managing the player data. If the application were to be improved to allow the player to have multiple clients (different devices, web clients, and so on), the data synchronization mechanism would need to be revisited and improved.

Uploading Settings Data to a Remote Server

In the previous chapter, you learned how to use the `AsyncTask` class to handle downloading of data from the network server. In this lesson, we turn our attention to uploading data, including the player settings and current score, to the network server. These goals are best achieved in the following order:

1. Begin with the settings screen and implement the functionality to generate a unique identifier for the player, and then upload and create a valid player record on the network server. This code is more complex than the simple network downloads of the previous chapter.
2. Once a valid player record with a unique identifier has been created, update the game screen to send score data up to the server at regular intervals. Because you have already generated an `AsyncTask` that routinely contacts the network server, adding this feature is trivial.
3. Update the scores screen to retrieve the appropriate player's friend data by changing the hard-coded player identifier to the one stored in the application preferences. This is a trivial change.

As you can see, creating the valid player record is vital for all the other features to work properly. It is also the most complex feature and the basis of most of our work in this chapter. In Chapter 10, “Collecting User Input,” you began creating the settings screen and storing application data in `SharedPreferences`. Now you will update the `QuizSettingsActivity` class to transmit the player settings to the server (in addition to storing it in `SharedPreferences`).

One fundamental difference between the networking features from earlier in this chapter and the creation of the settings screen is the workflow. For the game screen and the scores screen,

the user launched the appropriate activity, downloaded the data, and then used it. For the settings screen, the user launches the activity, sets any number of settings, which are immediately saved to the application preferences, and then we need to decide when it's the appropriate time to send all that information to the network server. We could send data piecemeal, each and every time there was a change, but that's not terribly efficient. Ideally, we want to send the data up to the server when the user is done entering any data he or she desires on the screen.

This is where things get tricky. You may be tempted to add a little button control to the screen to allow the user to initiate the network upload. This is not really the "Android" way. Instead, the user should be able to mosey on through the application at will. So, perhaps you might consider launching an asynchronous task when the activity is winding down, such as in the `onPause()` or `onDestroy()` methods within the activity lifecycle. You'd be on the right track. However, there's one problem: The `AsyncTask` class belongs to an `Activity` instance and is therefore bound by the `Activity` class lifecycle. In other words, when the activity goes away, you'd end up leaking the `AsyncTask`, because it cannot exist outside the activity.

What we really need is a way to spawn a background process that can live outside the activity. The Android SDK provides a mechanism for just this purpose, called an Android service.

Working with Android Services

A `Service` object is created by extending the `Service` class (`android.app.Service`) and defining the service within the `AndroidManifest.xml` file. The lifecycle of a `Service` is different from that of an `Activity` class. Because we need a simple service for the settings screen, we will not be exploring the complete lifecycle of a service. However, generally speaking, the `onCreate()` method is called, followed by either the `onStartCommand()` or `onBind()` methods, depending on the type of service and how it was started. When the service is finished, either because it completed and perhaps called the `stopSelf()` method or if there is no process bound to it, the `onDestroy()` method is called.

So, let's jump in and just create a service. What we want: a simple service to be used by the `QuizSettingsActivity` class to asynchronously upload player settings to the network server. For simplicity, we can give the service an `AsyncTask` class to encapsulate the networking code, much as we've done in previous occasions. The difference: The service will run the task as opposed to the activity. This way, the service might do other things in the future, should we desire it to.

To create a `Service` in the `QuizSettingsActivity` class, follow these steps:

1. Edit the `QuizSettingsActivity` class and add an inner class called `UploaderService`. This class should extend the `Service` class.
2. Create an inner class within the `UploaderService` called `UploadTask`. The `UploadTask` class should extend the `AsyncTask` class, much as you have seen in previous examples. This asynchronous task retrieves the application shared preferences, generates a unique identifier if the player record does not already exist, packages up the settings, and sends them off to the network server.

3. Give the `UploaderService` two member variables: a custom `DEBUG_TAG` for service-specific logging purposes and an `UploadTask` variable for the asynchronous task.
4. Override the `onStartCommand()` method of the `UploaderService` class. Have this method instantiate the `UploadTask` member variable and call the `execute()` method to start the task. Have this method return `START_REDELIVER_INTENT` so that the service will only remain running for the duration of its current task.
5. Override the `onBind()` method of the `UploaderService` class to return `null`. No binding is required for this simple service.

Once you implement the service, you must register it before it can be used. To register the service, update the Android manifest file for the project. The XML for this change would look like this:

```
<service android:name="QuizSettingsActivity$UploaderService"></service>
```

The resulting `UploaderService` class would look like this:

```
public static class UploaderService extends Service {
    private static final String DEBUG_TAG = "QuizSettingsActivity$UploaderService";
    private UploadTask uploader;

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        uploader = new UploadTask();
        uploader.execute();
        Log.d(DEBUG_TAG, "Settings and image upload requested");
        return START_REDELIVER_INTENT;
    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    private class UploadTask extends AsyncTask<Void, String, Boolean> {
        // UPLOADTASK IMPLEMENTATION HERE
    }
}
```

You can then launch the `UploaderService` in the `onPause()` method of the `QuizSettingsActivity` class, like this:

```
Intent uploadService = new Intent(getApplicationContext(), UploaderService.class);
startService(uploadService);
```


Implementing UploadTask

To communicate with the application server, you can leverage the `HttpClient` package (`org.apache.http`) included in the Android SDK. This package provides utilities for handling a wide variety of HTTP networking scenarios within your application. You can use `HttpGet` to post query variables in the same way a web-form submission would work, using the HTTP `GET` method; and you can use `HttpPost` to post form variables and upload the avatar graphic in the same way a web form might use the HTTP `POST` method.

The application server was written with HTML web forms in mind. In fact, the server was tested using a standard HTML form before the Android client was written. By developing a web client before the Android client, you ensure that the client/server communication protocols used are standard and cross-platform compatible. When you use this procedure, you know that any platform—including Android—that can handle web form-style `GET` and `POST` methods will be compatible with this application server. This way, the application can rely on the Apache HTTP libraries—primarily the `org.apache.http.client` package.

The `UploadTask` class is implemented much like the other asynchronous tasks that you already completed. It has a member variable of type `SharedPreferences` to access the application preferences. The `onPreExecute()` method retrieves the preferences. The bulk of the interesting part of the task involves the `doInBackground()` method, as usual. This method is the entry point for all the background work. Here, we call the `postSettingsToServer()` method to do as the method is named:

```
@Override
protected Boolean doInBackground(Void... params) {
    boolean result = postSettingsToServer();

    Log.d(DEBUG_TAG, "Done uploading settings ");
    return result;
}
```

Uploading Player Data with the HTTP `GET` Method

The primitive player data—that is the unique identifier, the nickname, the email, the password, the score, the gender, the birth date, and the favorite place—is submitted to the application server by using the HTTP `GET` method via the `HttpClient` and `HttpGet` classes. To enable this feature, follow these steps:

1. Begin by adding new shared preferences values to the `QuizActivity` class for the player's identifier and unique identifier. Also, add a definition for the network server URL for adding or editing player data on the server. This might look like the following.

```
public static final String GAME_PREFERENCES_PLAYER_ID = "playerId";
public static final String GAME_PREFERENCES_UNIQUE_ID = "serverId";
public static final String HYRT_SERVER_ACCOUNT_EDIT =
    HYRT_SERVER_BASE + "receive";
```

2. Retrieve the appropriate player settings from the application shared preferences.
3. Generate a unique identifier, if one has not been previously created and saved. Use the `UUID` class (`java.util.UUID`), as previously discussed. Save it to the preferences after creating it.
4. Package the settings in a vector (`java.util.Vector`) of name-value pairs for easy transmission. (The HTTP query variable encoder helper uses a list of `org.apache.http.NameValuePair` objects, rather than a key-value based dictionary, such as a hashtable, so we've opted to use a Vector here.)
5. Generate the appropriate URL for adding or editing settings data. You can use the `format()` method of the `URLEncodedUtils` class (`org.apache.http.client.utils.URLEncodedUtils`) to include the vector data.
6. Create an `HttpGet` (`org.apache.http.client.methods.HttpGet`) request object using the URL.
7. Create an `HttpClient` (`org.apache.http.client.HttpClient`) and execute the `HttpGet` request.

Again, this is not Android-specific code, but standard Java using the common Apache libraries. The resulting `postSettingsToServer()` method of the `UploadTask` class looks like this:

```
private boolean postSettingsToServer() {
    boolean succeeded = false;

    String uniqueId = mGameSettings.getString(
        GAME_PREFERENCES_UNIQUE_ID, null);
    Integer playerId = mGameSettings.getInt(
        GAME_PREFERENCES_PLAYER_ID, -1);
    String nickname = mGameSettings.getString(
        GAME_PREFERENCES_NICKNAME, "");
    String email = mGameSettings.getString(GAME_PREFERENCES_EMAIL,
        "");
    String password = mGameSettings.getString(
        GAME_PREFERENCES_PASSWORD, "");
    Integer score = mGameSettings
        .getInt(GAME_PREFERENCES_SCORE, -1);
    Integer gender = mGameSettings.getInt(GAME_PREFERENCES_GENDER,
        -1);
    Long birthdate = mGameSettings.getLong(GAME_PREFERENCES_DOB, 0);

    Vector<NameValuePair> vars = new Vector<NameValuePair>();

    if (uniqueId == null) {
        // If we don't have a unique id yet, create one and save it
        uniqueId = UUID.randomUUID().toString();
        Log.d(DEBUG_TAG, "Unique ID: " + uniqueId);
    }
}
```

```

        // save it in the prefs
        Editor editor = mGameSettings.edit();
        editor.putString(GAME_PREFERENCES_UNIQUE_ID, uniqueId);
        editor.commit();
    }

    // add the uniqueId to the request
    vars.add(new BasicNameValuePair("uniqueId", uniqueId));

    if (playerId != -1) {
        // otherwise, we use the playerId to update data
        vars.add(new BasicNameValuePair("updateId", playerId
            .toString()));
        // and we go ahead and push up the latest score
        vars.add(new BasicNameValuePair("score", score.toString()));
    }
    vars.add(new BasicNameValuePair("nickname", nickname));
    vars.add(new BasicNameValuePair("email", email));
    vars.add(new BasicNameValuePair("password", password));
    vars.add(new BasicNameValuePair("gender", gender.toString()));
    vars.add(new BasicNameValuePair("dob", birthdate.toString()));

    String url = HYRT_SERVER_ACCOUNT_EDIT + "?"
        + URLEncodedUtils.format(vars, null);

    HttpGet request = new HttpGet(url);

    try {

        ResponseHandler<String> responseHandler =
            new BasicResponseHandler();
        HttpClient client = new DefaultHttpClient();
        String responseBody = client.execute(request,
            responseHandler);

        if (responseBody != null && responseBody.length() > 0) {
            Integer resultId = Integer.parseInt(responseBody);
            Editor editor = mGameSettings.edit();
            editor.putInt(GAME_PREFERENCES_PLAYER_ID, resultId);
            editor.commit();
        }
        succeeded = true;
    } catch (ClientProtocolException e) {
        Log.e(DEBUG_TAG, "Failed to get playerId (protocol): ", e);
    } catch (IOException e) {

```

```

        Log.e(DEBUG_TAG, "Failed to get playerId (io): ", e);
    }
    return succeeded;
}

```

This is a straightforward implementation of an HTTP GET request. It is typically not a good idea to send sensitive data across networks in plain text, but it works for our simple example. There are equivalent classes for secure connections; consult the Java documentation for their usage.

That concludes the implementation of the Settings screen data upload service. Now you can move on to the finishing touches in the other activities.

Uploading Score Data to a Remote Server

To upload the player score information to the application server, you could add yet another `AsyncTask` subclass to the `QuizGameActivity` class. But, why not just update the existing `BookListDownloaderTask` to communicate the player's score to the application server each time the game downloads new books? This can help reduce latency and increase network efficiency. The only downside is that the score isn't updated at every answer and so may be slightly outdated.

The network server URL for retrieving new books could simply include three extra query variables: a Boolean value to turn on score updates (off by default), the player identifier, and the current score. The server would then update the user's score each time a new batch of books is requested, doing double duty. To make this small change, update the `doInBackground()` method of the `BookListDownloaderTask` class, as follows:

```

SharedPreferences settings =
    getSharedPreferences(GAME_PREFERENCES, Context.MODE_PRIVATE);
Integer playerId = settings.getInt(GAME_PREFERENCES_PLAYER_ID, -1);
if (playerId != -1) {
    Log.d(DEBUG_TAG, "Updating score");
    Integer score = settings.getInt(GAME_PREFERENCES_SCORE, -1);
    if (score != -1) {
        pathToBooks +=
            "&updateScore=yes&updateId="+playerId+"&score="+score;
    }
}
}

```

The code is added just after the URL string is created, but before it's used so it can be updated. (See the complete method implementation in the sample code that accompanies this book if you have further questions.)

Downloading Friends' Score Data

Now that you have implemented the player identifier, you'll also want to take a moment to update the `QuizScoresActivity` class to use the real identifier instead of the hardcoded one. Replace the hard-coded (2008) identifier with the following code:

```
SharedPreferences prefs = getSharedPreferences(GAME_PREFERENCES,
    Context.MODE_PRIVATE);
Integer playerId = prefs.getInt(GAME_PREFERENCES_PLAYER_ID, -1);
```

Summary

In this chapter, you modified the Have You Read That? application to download scores and book batches as well as upload game data—including player settings and their score—to a remote application server. You also learned how to create a background service that can run outside your application's activity lifecycle, when necessary. In addition, you learned how to use the HTTP GET method with the `HttpClient` class when uploading data to a server. You also learned about many of the issues to be aware of when developing network-enabled mobile applications. Think of this chapter as mastering the “building blocks” of networked applications.

Exercises

1. Test the Have You Read That? application in a variety of network situations using the emulator. Modify the emulator settings to simulate a slow network, and then run the application and view the results.
2. Test the Have You Read That? application in a variety of network situations using a device. Modify the device network settings (Airplane mode or try the cookie tin trick), and then run the application and view the results.
3. [Advanced] Modify the application to use the `Thread` and `Handler` methods for background processing instead of the `AsyncTask` method.
4. Modify the `BOOK_BATCH_SIZE` variable defined within the `QuizActivity` class. The question batches will be smaller, and thus retrieved more frequently, but the score data will be uploaded more often.
5. Override the other callback methods of the `UploaderServer` class, such as `onDestroy()`, and add informational log messages to each.

Exploring the Amazon Web Services SDK for Android

The Amazon Web Services SDK for Android is a library available for download that provides access to Amazon Web Services (AWS) through a series of APIs. AWS are Amazon's set of services for creating scalable cloud-based applications. In this chapter, we explore what services are available in this SDK and how they might be used with mobile applications for the Android platform.

AWS Is Not Just for the Amazon Kindle Fire

The AWS SDK for Android can be used with all Android devices running API Level 7 (Android 2.1) and higher, not just Kindle Fire devices. The contents of this chapter apply to all such compatible devices.

The 10,000-Foot View of AWS

As a major web presence, Amazon has developed its own cloud-computing services for developers to leverage—the same services are used by the Amazon.com retail website to make it the robust, scalable retail venue with which we are all familiar. Leveraging AWS usually requires the developer to create and configure a developer account associated with the service. Fees are charged based on the usage of such services, usually on a sliding scale—often with a free tier for minimal usage. Some services have a free tier, either for the first year or indefinitely.

The beauty of AWS is that its services, such as cloud storage or cloud computation, scale well with reasonable fee scales, which make them ideal for use even by the developer of the smallest niche app or the corporation serving to millions of users. And if your niche app goes viral and you suddenly have millions of users, AWS can keep up.

AWS has been available to web applications for some time. Android developers can also easily leverage the services by using the AWS SDK for Android. The SDK includes

- A library for accessing the AWS services
- Code samples that illustrate how to use the AWS services with Android apps
- Java documentation (Javadoc) to accompany the library

Developers can use the developer sandbox to test service usage before accessing production services in order to iron out any application glitches. Now, let's look at the services available in more detail and discuss how they might be used with Android applications.

Exploring the AWS Offerings

A variety of different services are available through the AWS SDK for Android. Database and storage services include

- Amazon DynamoDB
- Amazon SimpleDB
- Amazon S3

Messaging and notification services include

- Amazon Simple Email Service (SES)
- Amazon Simple Notification Service (SNS)
- Amazon Simple Queue Service (SQS)

Infrastructure and administrative services include

- Amazon EC2
- Amazon Elastic Load Balancing
- Amazon CloudWatch
- Amazon Auto Scaling

All AWS are protected by sophisticated authentication mechanisms. An authentication API provides interfaces to several of these methods. For services that would require the user's keys and secrets, such as Amazon EC2, an API for giving these credentials is available. For other services that might need your credentials, such as for securely accessing data in SimpleDB, a mechanism is provided to securely allow the user's access without revealing your own secret credentials. This allows your mobile applications to access your backend resources and bill you for the backend resource charges instead of the end user.

Using AWS Database and Storage Services

Amazon provides a number of simple, scalable database and storage services for your convenience. Generally speaking, these services allow developers to enable their applications to store data persistently and securely in the cloud. No longer does the developer need to worry if his application server database can scale—AWS does the scaling for you, and it's affordable, provided you price your application such that the fees are easily covered.

Amazon DynamoDB is a NoSQL-style database that allows you to create a table in the cloud. It is not a relational database, but simple structured storage that is accessed via an API. Android applications that need to store simple user data—like preferences or settings or basic account information—might want to use this highly efficient “flat” storage service.

Amazon SimpleDB is a nonrelational database that uses keys and values to store and retrieve data efficiently. This is not structured storage, but it is well suited for developers who need to store and quickly access data without having to generate a schema or manage database operations. Android applications might use this service to keep track of global user information, like high scores or (average) ratings for movies, books, or music content.

Amazon S3 is Amazon's most popular and robust storage service. S3 stands for Simple Storage Service. This service allows developers to store and retrieve data of nearly any size, up to 5 terabytes—basically a “blob” storage service. Storage fees are based on how much content you store. Android developers might use S3 to serve up wallpaper content, new game levels, or other application assets that could be dynamically applied. Developers could also use S3 to provide users with their own cloud-based storage; for example, users might upload photos or music and have it accessible in the “cloud” from multiple devices (such as the cloud-based music player).

Using AWS Messaging and Notification Services

Amazon provides some simple messaging and notification services that your applications can leverage so you do not have to produce your own scalable services. Pricing for messaging and notification services is often a combination of per-message charges as well as data transfer-based tiers.

Amazon Simple Email Service (SES) is an email service that developers can use instead of having to host an email server. Messages can be sent securely, in bulk, with attachments and such. Amazon SES is also designed to avoid some of the common pitfalls that bulk email servers suffer from (automatically identified as spammers) and play nice with Internet service providers (ISPs) so that your messages get delivered where they should. Android applications might use this service to enable a feature that allows the app user to send content, such as a newspaper article or blog post, to a friend.

The Amazon Simple Notification Service (SNS) is a broader notification service that acts as a push service for various notification protocols, like SMS, email, or HTTP. Android applications might use this service to send friend requests or game challenges between users via SMS. There's no need for any application server infrastructure to manage push.

The Amazon Simple Queue Service (SQS) is a queue that can store messages persistently, allowing developers to create sophisticated workflows. Android applications might use this service to manage more complicated cloud processing of data that cannot reasonably be completed on the Android device.

Using AWS Infrastructure and Administrative Services

In addition to client-oriented services, the AWS SDK for Android also includes APIs for administrative-oriented tasks. Although these services would not likely be built into a game or photo application per se, they would be useful in tools and utilities for direct customers of AWS. These APIs are more appropriate for the administrator of the server, not directly by the client Android application.

Amazon EC2 is the most generic of all Amazon's cloud services. It provides the ability to provision and boot web servers in a matter of minutes in nearly any quantity with capabilities ranging from the simple to the very high end, with prices ranging from as little as two cents an hour to as much as many dollars an hour for high-end servers. Amazon Elastic Load Balancing is a service for distributing incoming requests across numerous EC2 instances. Android applications might use EC2 as an application server.

Amazon CloudWatch is a cloud-monitoring solution that can monitor the health of your cloud resources. Various metrics can be monitored across several of Amazon's web services, including EC2 instances, load balancing, SQS, SNS, and even custom metrics. Metrics can be used for controlling Amazon Auto Scaling, which allows you to scale your EC2 instances as your needs change.

Summary

Android applications running Android 2.1 and higher can leverage AWS through the SDK made available for Android developers. The SDK includes a number of cloud-based services that allow developers to scale their applications smoothly and on a sliding fee scale. AWS offerings include database and data-storage services, messaging and notification services, and infrastructure and administrative services. Finally, you learned about how the SDK is componentized so that developers can include only those services needed by their application and not unnecessarily bloat the application package file.

Exercises

1. Review the AWS service offerings. List three ways that you could incorporate these services into the Have You Read That? application.
2. Download the AWS SDK for Android from <http://aws.amazon.com/sdkforandroid/>.

3. Browse through the AWS SDK for Android documentation. Read the Getting Started Guide online at <http://aws.amazon.com/sdkforandroid/>.
4. Browse through the AWS SDK for Android code samples. Choose one and load it up in Eclipse by creating a new Android project and choosing to create it from existing source code. Run the application and see what happens.
5. Check out the website <http://www.amazonappstoredev.com> for tutorials on using the AWS SDK for Android.

This page intentionally left blank



Publishing Your Kindle Fire Application

- 15** Managing Alternative and Localized Resources 233
- 16** Testing Kindle Fire Applications 249
- 17** Registering as an Amazon Application Developer 261
- 18** Publishing Applications on the Amazon Appstore 271

This page intentionally left blank

Managing Alternative and Localized Resources

The Android platform supports a very flexible resource system. By storing project resources, such as strings and graphics, in a certain, structured way, developers can have their applications load different resources under different conditions. For example, you may have noticed that the screens of the Have You Read That? application don't always look right if you hold your Kindle Fire in landscape mode. In this chapter, we discuss how to use alternative resources to support different screen modes, like portrait versus landscape, as well as how to manage resources for internationalization purposes.

Using the Alternative Resource Hierarchy

Few application user interfaces look perfect on every device and screen orientation. Most require some tweaking and special-case handling, and developing for devices like the Amazon Kindle Fire is no different.

The Android platform allows you to organize your project resources so that you can tailor your applications to specific criteria about the device state. It can be useful to think of the resources stored at the top of the resource hierarchy naming scheme as *default resources* and the specialized versions of those resources as *alternative resources*. Here are some reasons you might want to include alternative resources within your application:

- To support different devices with different screen types and sizes
- To support different device orientations, like portrait and landscape modes
- To support different user languages and locales
- To support different device docking modes

Understanding How Resources Are Resolved

Here's how it works: Each time a resource is requested within an Android application, the Android operating system attempts to match the best possible resource for the job. In many cases, applications provide only one set of resources—the default resources. However, developers have another option. They can include alternative versions of those same resources as part of their application packages. The Android operating system always attempts to load the most specific resources available—developers do not have to worry about determining which resources to load, because the operating system handles this task.

There are four important rules to remember when creating alternative resources:

- The Android platform always loads the most specific, most appropriate resource available. If an alternative resource does not exist, the default resource is used. Therefore, it's important to know your target devices, design for the defaults, and add alternative resources judiciously in order to keep your projects manageable.
- Alternative resources must always be named exactly the same as the default resources and stored in the appropriately named directory, as dictated by a special alternative resource qualifier. If a string is called `strHelpText` in the `/res/values/strings.xml` file, it must be named the same in the `/res/values-fr/strings.xml` (French) and `/res/values-zh/strings.xml` (Chinese) string files. The same goes for all other types of resources, such as graphics or layout files.
- Good application design dictates that alternative resources should almost always have a default counterpart so that, regardless of the device configuration, some version of the resource will always load. The only time you can get away without a default resource is when you provide every kind of alternative resource. One of the first steps the system takes when finding a best-matching resource is to eliminate resources that are contradictory to the current configuration. For example, in portrait mode, the system would not even attempt to use a landscape resource, even if that is the only resource available. Keep in mind that new alternative resource qualifiers are added over time, so although you might think your application provides complete coverage of all alternatives now, it might not in the future.
- Don't go overboard creating alternative resources, because they add to the size of your application package and can have performance implications. Instead, try to design your default resources to be flexible and scalable. For example, a good layout design can often support both landscape and portrait modes seamlessly—if you use appropriate layouts, user interface controls, and scalable graphic resources.

Organizing Alternative Resources with Qualifiers

Alternative resources can be created for many different criteria, including, but not limited to, screen characteristics, device input methods, and language or regional differences. These alternative resources are organized hierarchically within the `/res` resource project directory. You use directory qualifiers (in the form of directory name suffixes) to specify a resource as an alternative resource to load in specific situations.

A simple example might help drive this concept home. The most common example of when alternative resources are used has to do with the default application icon resources created as part of a new Android project in Eclipse. An application could simply provide a single application icon graphic resource, stored in the `/res/drawable` directory. However, different Android devices have different screen resolutions. Therefore, alternative resources are used instead: `/res/drawable-hdpi/icon.png` is an application icon suitable for high-density screens, `/res/drawable-ldpi/icon.png` is the application icon suitable for low-density screens, and so on. Note that, in each case, the alternative resource is named the same. This is important. Alternative resources must use the same names as the default resources. This is how the Android system can match the appropriate resource to load—by its name.

Here are some additional important facts about alternative resources:

- Alternative resource directory qualifiers are always applied to the default resource directory name (for example, `/res/drawable-qualifier`, `/res/values-qualifier`, `/res/layout-qualifier`).
- Alternative resource-directory qualifiers (and resource filenames) must always be lowercase, with one exception: region qualifiers.
- Only one directory qualifier of a given type may be included in a resource directory name. Sometimes, this has unfortunate consequences—you might be forced to include the same resource in multiple directories. For example, you cannot create an alternative resource directory called `/res/drawable-ldpi-mdpi` to share the same icon graphic. Instead, you must create two directories: `/res/drawable-ldpi` and `/res/drawable-mdpi`. Frankly, when you want different qualifiers to share resources instead of providing two copies of the same resource, you're often better off making those your default resources and then providing alternative resources for those that do not match `ldpi` and `mdpi`—that is, `hdpi`. As we said, it's up to you how you go about organizing your resources; these are just our suggestions for keeping things under control.
- Alternative resource directory qualifiers can be combined or chained, with each qualifier being separated by a dash. This enables developers to create specific directory names and, therefore, very specialized alternative resources. These qualifiers must be applied in a specific order, and the Android operating system always attempts to load the most specific resource (that is, the resource with the longest matching path). For example, you can create an alternative resource directory for French language (qualifier `fr`), Canadian region (qualifier `rCA`—this is a region qualifier and is therefore capitalized) string resources (stored in the values directory) as follows: `/res/values-fr-rCA/strings.xml`.
- You only need to create alternative resources for the specific resources you require—not every resource in your application. If you only need to translate half the strings in the default `strings.xml` file, only provide alternative strings for those specific string resources. In other words, the default `strings.xml` resource file might contain a superset of string resources with the alternative string resource files containing a subset—only the strings requiring translation. Common examples of strings that do not get localized are company or brand names.

- No custom directory names or qualifiers are allowed. You may only use the qualifiers defined as part of the Android SDK.
- Always try to include default resources—that is, those resources saved in directories without any qualifiers. These are the resources that the Android operating system will fall back on when no specific alternative resource matches the criteria. If you don't include these, the system falls back on the closest matching resource based on the directory qualifiers—one that might not make sense.

Now that you understand how alternative resources work, let's look at some of the directory qualifiers that you can use to store alternative resources for different purposes. Qualifiers are tacked on to the existing resource directory name in a strict order, as described in the Android SDK documentation, which is available at <http://goo.gl/e7dfv>.

Using Alternative Resources Programmatically

There is currently no easy way to request resources of a specific configuration programmatically. For example, the developer cannot programmatically request the French or English version of the string resource. Instead, the Android system determines the resource at runtime, and developers refer only to the general resource variable name.

Organizing Application Resources Efficiently

It's easy to go too far with alternative resources. You could provide custom graphics for every different permutation of device screen, language, or input method. However, each time you include an application resource in your project, the size of your application package grows.

There are also performance issues with swapping out resources too frequently—generally when runtime configuration transitions occur. Each time a runtime event, such as an orientation change, occurs, the Android operating system restarts the underlying `Activity` and reloads the resources. If your application is loading a lot of resources and content, these changes come at a cost to application performance and responsiveness.

Choose your resource organization scheme carefully. Generally, you should put the most commonly used resources as your defaults and then carefully overlay alternative resources only when necessary. For example, if you are writing an application that routinely shows videos or displays a game screen, you might want to make landscape-mode resources your defaults and provide alternative portrait-mode resources because they are not as likely to be used.

If your application is only targeting the Amazon Kindle Fire, only certain types of qualifiers are likely to apply to your application. You will need a good set of default resources, and you will likely use the orientation qualifiers if your application is to support both portrait and landscape modes, but you will not have to clutter your application with a lot of other qualifiers that are meant to be used by applications that run on a variety of Android devices.

Customizing the Application Experience

Let's look at a simple example of how to customize the application experience for different device configurations. The most common situation for Kindle Fire applications is that users often change the device orientation between portrait and landscape modes. Therefore, let's look at each screen of the Have You Read That? application and make some tweaks so the experience is pleasant, regardless of the device orientation.

Updating the Main Screen

The main screen, if you recall, uses a `GridView` control to hold the various icons that users can tap to go to different features of the game. The `GridView` control looks fine in portrait mode, but what happens in landscape mode? Well, it's not pretty. Figure 15.1 shows you what we're talking about; the second row of game options is clipped.

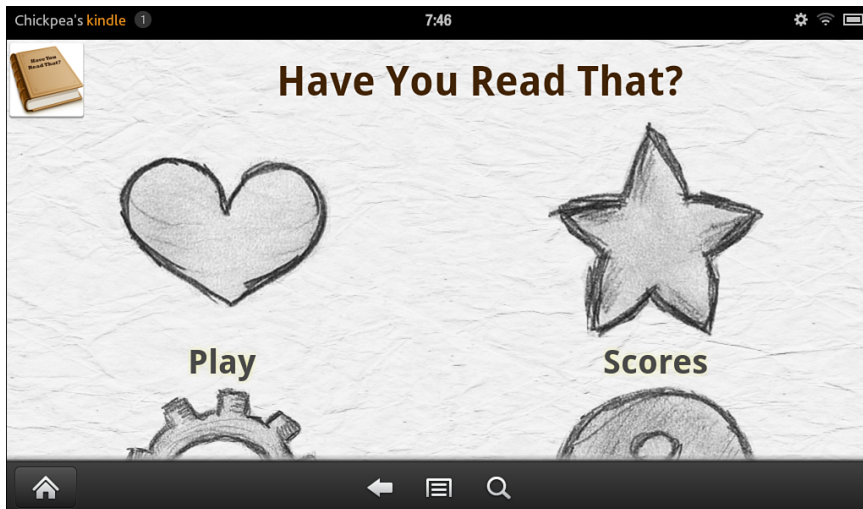


Figure 15.1 Main Screen in Landscape Mode, But Broken

So, what can we do to fix this problem? Well, there are several options. One option is to just make the game portrait-mode only all the time. However, users tend to like using landscape mode on their Kindle Fire devices, so why take this away from them when you can easily update the application to support both orientations? Another option is to scale the graphics, making them smaller so they fit vertically or horizontally in a 2×2 grid. Not the best solution, but it's an easy option.

Another solution that is more elegant is to adjust the number of columns in the `GridView` control based on the orientation of the device. To do this, create a copy of the `menu.xml` file and place it in a new directory called `layout-land`. This is where your landscape-mode layout

resource files will go. Android will load the correct file based on the orientation of the device. Inside the landscape version of the file, set the `numColumns` value to 4, whereas in the default version of the resource, the `numColumns` value is set to 2. Now, when you run the application and rotate the screen into portrait mode, you see a screen like Figure 15.2. This is much improved!

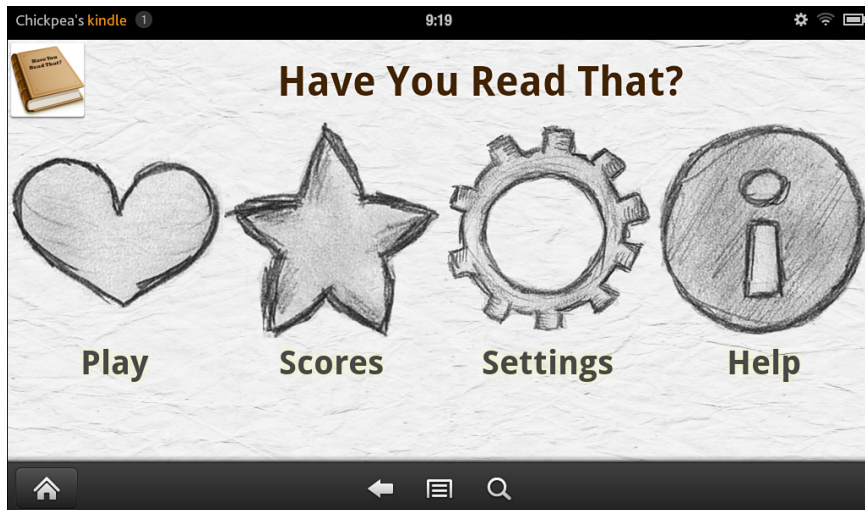


Figure 15.2 Main Screen in Landscape Mode, Now Fixed

Updating the Splash Screen

The splash screen also suffers from the images and text not showing completely in landscape mode. In addition, the animation doesn't look quite right. This time, we'll fix the splash screen by just modifying the existing layout to make it work in both orientations. The animation, however, will need to be customized for the different orientations.

There are three problems to solve for landscape mode. First, the bottom text, "Read That?" is clipped. Second, the images are clipped. Finally, the start of the animation isn't off the screen. All these issues can be fixed without adding an alternative layout resource. Instead, we will modify the dimensional sizes used by the layout.

You've seen dimension resource types used. Dimension resources allow you to configure reusable values. In this case, we create different values, with the same dimension resource names, for large landscape devices like Kindle Fire. (The size qualifiers like `small` and `large` are relative to phone screens, where `large` is usually a 7" device and `xlarge` a 10" device.) To do this, create a new folder called `values-large-land`. This will hold resource values (such as dimensions) for large screens (as found on the Kindle Fire and as used with the other dimensions, as you may have seen in the code) for landscape orientation. In it, create a new `dimens.xml` file. Within this file, we'll put in new values for the four dimensions referenced in the splash screen layout

resource. All four values are too large for landscape mode, so reduce them to lower values, such as

```
<dimen name="splash_title_size">48dp</dimen>
<dimen name="splash_version_size">15dp</dimen>
<dimen name="splash_version_spacing">3dp</dimen>
<dimen name="splash_books_size">290dp</dimen>
```

This solves the first two problems. The last problem is the animation. It doesn't start out off the screen now. The solution is similar. Create a new folder called `anim-land` and put modified versions of the `slide_in_1.xml` and `slide_in_2.xml` animation resources. Modify the animations to work properly in landscape mode. They could do the same sort of animation or change the animation sequence entirely. It's up to you.

Updating the Game Screen

The game screen requires changes that are similar to those of the splash screen, though even less needs to be fixed. In fact, this is an example of a screen that simply needs a slightly updated design to work well in both screen orientations.

The problem with this screen is that the book title is pushed off the bottom edge of the screen. Additionally, some of the text could be larger for readability, because there's a lot of screen real estate available in landscape mode. So, for example, we can provide a different layout resource that places the text in the foreground and the image as the background. With some tweaking to the `game_book_title_text_size` and `game_book_header_text_size` dimension values in the `values-large-land` folder, the game screen will now look like what's shown in Figure 15.3.

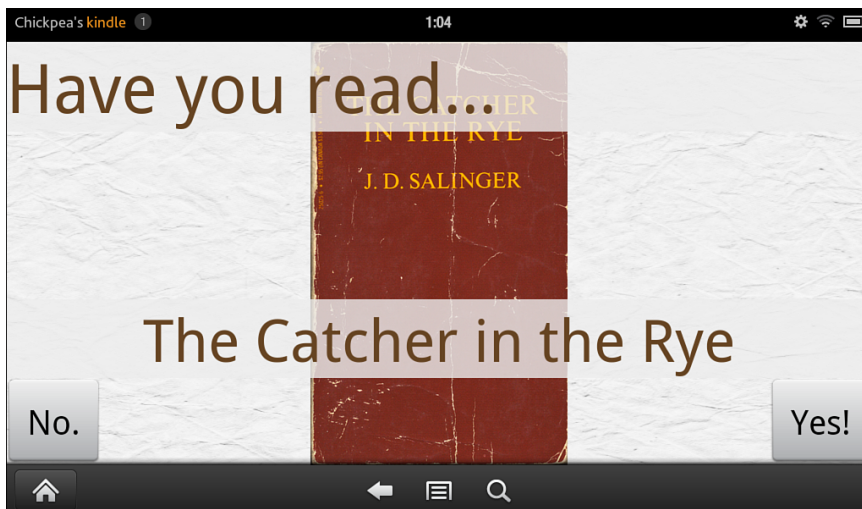


Figure 15.3 Game Screen in Landscape Mode, Updated

Updating the Other Screens

The help screen, our final screen that needs some tweaking, is again a matter of a design flaw that is easily fixed to work on both orientations. For this screen, simply wrap the `TextView` inside a `ScrollView` so that when the screen is not long enough to display the text, the user can scroll down to see more!

Internationalizing Android Applications

With a global marketplace, developers can maximize profits and grow their user base by supporting a variety of different languages and locales. Let's take a moment to clarify some terms. Although you likely know what we mean by *language*, you may not be aware that each language may have a number of different locales. For example, the Spanish spoken in Spain is different from that spoken in the Americas; the French spoken in Canada differs from that spoken in Europe and Africa; and the English spoken in the United States differs from the English spoken in Britain. English is a *language*, while English (United States), English (United Kingdom), and English (Australia) are *locales* (see Figure 15.4).

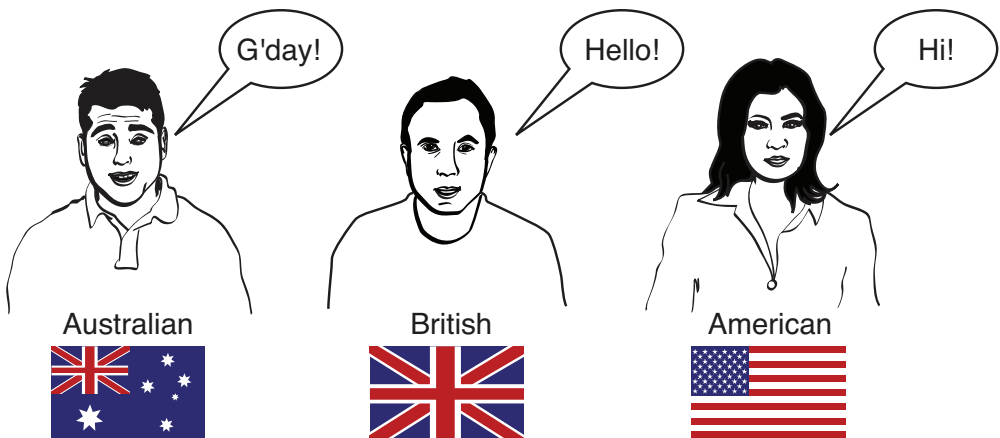


Figure 15.4 People Who Speak the Same Language Often Have Localized Dialects

Applications are made up of data and functions (behavior). For most applications, the behavior is the same, regardless of the locale. However, the data must be localized. This is one of the key reasons resource files exist—to externalize application data.

Locale and language differences go far beyond “accents”—to include different word spellings, meanings, slang, and format of regional data, like date and time and primary currency. The most common type of application data that requires localization is the strings of text used by the application. For example, a string of data might represent a user's name, but the text label for that value on an application screen would need to be shown in the proper language (for example, Name, Nom, Nombre).

Development platforms that support internationalization typically allow for string tables, which can be swapped around so that the same application can target different languages. The Android platform is no exception. Do not hard-code localizable data, like string information, into the application source files—Java and layout-resource files especially—unless absolutely necessary. Doing so hinders internationalization efforts, especially when targeting different devices.

How Android Localization Works

Compared to other mobile platforms, the Android SDK provides extensive support for internationalization using alternative resources. However, at this time, Kindle Fire does not support locale or language changes by users at the system level.

Android localization considerations fall into three main categories:

- The languages and locales supported by the Android platform (an extensive list—the superset of all available languages).
- The languages and locales supported by a specific Android device. (At this time, Kindle Fire only supports English and is distributed in the United States.)
- The countries, languages, and locales supported by the distribution mechanism. (Amazon Appstore is currently only available in the United States.)

New locales are added with each new Android SDK, so for a complete list of the locales supported for a given Android SDK, see the specific platform documentation. For example, the Android API Level 10 locale support is listed here: <http://goo.gl/wJK5N>.

How the Android Operating System Handles Locale

Much like other operating systems, the Android platform has a system setting for locale. This setting has a default that can be modified by the device manufacturer or carrier. For example, a German mobile operator might make the default locale Deutsch (Deutschland) for its shipping devices. Amazon Kindle Fire is currently available only in the United States, so its locale is set as such to English (American).

On many Android devices, a user can change the system-wide setting for locale in the system settings. That said, at this time, there is no exposed setting on Kindle Fire.

The system setting affects the behavior of applications installed on the device.

How Applications Handle Locales

Now, let's look at how the system-wide locale setting affects each Android application. When an Android application uses a project resource, the Android operating system attempts to match the best possible resource for the job at runtime. In many cases, this means checking for

a resource in the specific language or regional locale. If no resource matches the required locale, the system falls back on the default resource.

Developers can specify specific language and locale resources by providing resources in specially named resource directories of the project. Any application resource can be localized, whether it is a string resource file, a drawable, an animation sequence, or some other type.

Specifying Default Resources

Default resources are the most important resources, because they are the fallback for any situation when a specific, tailored resource does not exist (which happens more often than not). In the case of the Have You Read That? application, all the default resources are in English, but they need not be.

Specifying Language-Specific Resources

To specify strings for a specific language, you must supply the resource under a specially named resource directory that includes the two-letter language code provided in ISO 639-1 (see <http://goo.gl/ToTSo>). For example, English is `en`, French is `fr`, and German is `de`. Let's look at an example of how this works.

Say that you want the Have You Read That? application to support English, German, and French strings. You would take the following steps:

1. Create a `strings.xml` resource file for each language. Each string that is to be localized must appear in each resource file with the same name, so it will be programmatically loaded correctly. Any strings that you don't want to localize can be left in the default (English) `/res/values/strings.xml` file.
2. Save the French `strings.xml` resource file to the `/res/values-fr/` directory.
3. Save the German `strings.xml` resource file to the `/res/values-de/` directory.

Android can now grab the appropriate string, based on the system locale, at runtime and load it. However, if no match exists, the system falls back on the default string file defined in the `/res/values/` directory. This means that if English (or Arabic, or Chinese, or Japanese, or an unexpected locale) is chosen, the default (fallback) English strings will be used.

Similarly, you could provide German-specific drawable resources to override the default graphics in the `/res/drawable/` directory by supplying versions (each with the same name) in the `/res/drawable-de/` directory.

Specifying Region-Specific Resources

You may have noticed that the previous example specifies high-level language settings only (English, but not American English versus British English versus Australian English). Don't worry! You can specify the region or locale as part of the resource directory name as well.

To specify strings for a specific language and locale, you must store the localized resource under a specially named directory that includes the two-letter language code provided in ISO 639-1 (see <http://goo.gl/ToTSo>), followed by a dash, then a lowercase `r`, and finally, the ISO 3166-1-alpha-2 region code (see <http://goo.gl/EyyJv>). For example, American English is `en-rUS`, British English is `en-rGB`, and Australian English is `en-rAU`. Let's look at an example of how this works.

If you want the Have You Read That? application to support these three versions of English, you could do the following:

1. Create a `strings.xml` resource file for each language. Any strings you don't want to localize can be left in the default (American English) `/res/values/strings.xml` file.
2. Save the British English `strings.xml` resource file to the `/res/values-en-rGB/` directory.
3. Save the Australian English `strings.xml` resource file to the `/res/values-en-rAU/` directory.

To summarize, start with a default set of resources—which should be in the most common language your application will rely on. Then, add exceptions—such as separate language and region string values—where needed. This way, you can optimize your application so it runs on a variety of platforms. For a more complete explanation of how the Android operating system resolves resources, check out the Android developer website (<http://goo.gl/qUP0h>).

How Kindle Fire Handles Locales

At the time of this writing, Kindle Fire is only available in the United States. The device has no accessible language or locale configuration. Most Android devices have language settings that allow the user to change the system settings. It is possible that a popular device, like Amazon's Kindle Fire, would be internationalized in the future.

Since the user cannot change the system locale on the device, the only way to create international language applications is by putting them as default resources or by dynamically loading them outside of the alternative resource system. Note that even internationalized applications must have their Amazon Appstore product description written in English at this time. Therefore, it may be a bit premature to bother providing alternative locale and regional resources if your application will only target the Kindle Fire.

Android Internationalization Strategies

Don't be overwhelmed by the permutations available to developers when it comes to internationalizing an application. Instead, give some thought to how important internationalization is to your application during the design phase of your project. Develop a strategy that suits your specific needs and stick to it.

Here are some basic strategies to handle Android application internationalization:

- Forgo internationalization entirely for now because the Amazon Appstore is not an international marketplace (yet).
- Limit internationalization to a few target languages your application is likely to benefit from having. Make alternative versions of your application for each language or load language resources dynamically.
- Implement full internationalization for target audiences using alternative resources, assuming that most devices (if not the Kindle Fire) will support this strategy.

Now let's talk about each of these strategies in more detail.

Forgoing Application Internationalization

Whenever possible, save your development and testing teams a lot of work—don't bother to internationalize your application. This is the “one size fits most” approach to mobile development, and it is often possible with simple, graphic-intensive applications like games that do not have a lot of text to display. If your application is simple enough to work smoothly with internationally recognized graphical icons (such as play, pause, stop, and so on) instead of text labels, or “Sims” language (garbled mumbles that get the point across to speakers of any language), you may be able to forgo internationalization entirely. Games like tic-tac-toe and chess are games that require little, if any, text resources.

Some of the pros of this strategy are the following:

- Development and testing are simplified.
- Amazon Appstore is not internationalized at this time.
- Amazon Kindle Fire has no locale or language settings at this time.
- The application size is the smallest (only one set of resources).

Some of the cons of this strategy are the following:

- For text- or culture-dependant applications, this approach greatly reduces the value of the application. It is simply too generic.
- This strategy automatically alienates certain audiences and limits your application's potential marketplaces.

This technique works only for a subset of applications. If your application requires a help screen, for example, you're likely going to need at least some localization for your application to work well all over the world.

Limiting Application Internationalization

Most applications require only some light internationalization. This often means internationalizing string resources only, but other resources, such as layouts and graphics, remain the same for all languages and locales.

Some of the pros of this strategy are the following:

- Modest development and testing requirements
- Streamlined application size (specialized resources kept to a minimum)

Some of the cons of this strategy are the following:

- Application may still be too generic for certain types of applications. Overall design (especially screen design) may suffer from needing to support multiple target languages. For example, text fields might need to be large enough to support verbose languages, such as German, but look odd and waste valuable screen real estate in less verbose languages.
- At this time, you cannot provide internationalized applications for Kindle Fire using alternative resources; you must load your internationalized data programmatically and work outside the resource system to do so.
- Because you've headed down the road of providing language-specific resources, your users are more likely to expect other languages you haven't supported. In other words, you're more likely to start getting requests for your app to support more languages if you've supported some. That said, you've already built and tested your application on a variety of languages, so adding new ones should be straightforward.

Implementing Full Application Internationalization

Some types of applications require complete internationalization. Providing custom resources for each supported language and locale is a time-intensive endeavor, and you should not do it unless you have a really good reason to do so, because the size of the application grows as you include more data and resources. This approach often necessitates breaking the individual languages into separate APK files for publication, resulting in more complex configuration management. However, this allows a developer to tailor an application for each specific marketplace to a fine degree.

Some of the pros of this strategy are the following:

- The application is fully tailored and customized to individual audiences; this strategy allows for tweaks to individual locales.
- It builds user loyalty by providing users with the best, most customized experience.
- You can reach international users before others do, because the alternative resources method will not currently work with Kindle Fire, so few developers will bother to internationalize their applications for this device.

Some of the cons of this strategy are the following:

- It is the most lengthy and complicated strategy to develop.
- Each internationalized version of the application must be fully tested as if it were a completely different application (which it may well be, if you are forced to split it into different APK files due to application size).

Beware of over-internationalizing an application. The application package size will grow as you add language- and locale-specific resources—whether you use the alternative resource method or build your own solution. There is no reason to head down this road unless you have a compelling reason to do so—and unless you have the development, testing, and product team to manage it. Having a poorly localized version of an application can be worse to your image than having no localization at all.

Using Localization Utilities

The Android SDK includes support for handling locale information. For example, the `Locale` class (`java.util.Locale`) encapsulates locale information.

Determining System Locale

If you need to modify application behavior based on locale information, you need to be able to access information about the Android operating system. You can do this by using the `getConfiguration()` method of the `Context` object, as follows:

```
Configuration sysConfig = getResources().getConfiguration();
```

One of the settings available in the `Configuration` object is the locale:

```
Locale curLocale = sysConfig.locale;
```

You can use this locale information to vary application behavior programmatically, as needed.

Formatting Strings Like Dates and Times

Another aspect of internationalization is displaying data in the appropriate way. For example, U.S. dates are formatted MM/DD/YY and October 27, 2011, whereas much of the rest of the world uses the formats DD/MM/YY and 27 October 2011. The Android SDK includes a number of locale-specific utilities. For example, you can use the `DateFormat` class (`android.text.format.DateFormat`) to generate date and time strings in the current locale, or you can customize date and time information as needed for your application. You can use the `TimeUtils` class (`android.util.TimeUtils`) to determine the time zone of a specified country by name.

Handling Currencies

Much like dates and times, currencies and how they are formatted differ by locale. You can use the standard Java `Currency` class (`java.util.Currency`) to encapsulate currency information. Use the `NumberFormat` class (`java.text.NumberFormat`) to format and parse numbers based on locale information.

Summary

In this chapter, you learned how to strategically use alternative resources to support a variety of Kindle Fire device configurations, such as different orientations, by modifying the *Have You Read That?* application to support both portrait and landscape mode in a reasonable fashion. You also learned how the Android platform handles different countries, languages, and locales. Finally, you learned how to organize Android application resources using a number of different internationalization strategies.

Exercises

1. Add a new set of string resource values to the *Have You Read That?* application in the language or locale of your choice. Test the results in the Android emulator.
2. Change the *Have You Read That?* application so that it loads a different drawable or color resource for portrait versus landscape modes. For example, change the paper graphic on the main menu to something more specific to that language/locale. Test the results in the Android emulator or your Kindle Fire.
3. Review the qualifier types listed in the table in the alternative resources Android SDK documentation (<http://goo.gl/E8v1d>).

This page intentionally left blank

Testing Kindle Fire Applications

Every mobile developer dreams of developing a “killer app.” Many people think that if they could just come up with a great idea, success is guaranteed. This is, unfortunately, not the case. The truth is, people come up with great ideas all the time. The trick is to act on the idea with a clear vision, a concise “pitch” to users, and an intuitive user interface. Applications should also be carefully paired with the devices they target; the Amazon Kindle Fire is primarily an e-book reader with supplementary applications whose features should be appropriate for the device features as well as its audience. There’s also a time component: You have to get that app into users’ hands quickly—before someone else does! A killer app must have the right mix of these ingredients, but a poor implementation of an excellent idea isn’t going to become a killer app, so it’s important to test each application thoroughly before publication. In this chapter, you learn how to test mobile applications in a variety of ways.

Testing Best Practices

Mobile users expect a lot from today’s mobile applications. They expect the applications they install to be stable, responsive, and secure. *Stable* means that the application works and doesn’t crash or mess up the user’s device. *Responsive* means that the device always responds to button presses and tap events, and long operations use progress bars or other forms of activity indicators. *Secure* means that the application doesn’t abuse the trust of the user, either intentionally or unintentionally. Users expect an application to have a reasonably straightforward user interface, and they expect the application to work 24 hours a day, seven days a week, especially when it comes to networked applications.

It might seem like users expect a lot for an application that might be priced at \$0.99, but really, do any of these expectations seem that unreasonable? We don’t think so. However, they do impose significant responsibilities on a developer in terms of testing and quality control.

Whether you're a project team of one or one hundred, every mobile development project benefits from a good development process with a solid test plan. The following are some quality measures that can greatly improve the development process:

- Coding standards and guidelines
- Regular versioned builds
- A defect tracking system with a process for resolving defects
- Systematic application testing using a test plan

Because of the speed at which mobile projects tend to progress, iterative development processes are generally the most successful strategies for mobile development. Rapid prototyping gives developers and quality assurance personnel ample opportunities to evaluate an application before it reaches users.

Developing Coding Standards

When developers have and follow a set of predetermined guidelines, their code is more cohesive, easier to read, and easier to maintain. Developing a set of well-communicated coding standards for developers can help drive home some of the important requirements of the mobile applications we've been discussing. For example, developers should

- Discuss and come up with a common way for all developers to implement error and exception handling.
- Move lengthy or process-intensive operations off the main UI thread.
- Release objects and resources that aren't actively being used.
- Practice prudent memory management and track down memory leaks.
- Use project resources appropriately. For example, don't hard-code data and strings in code or layout files.

Performing Regular Versioned Builds

Implementing a reproducible build process is essential for a successful Android project. This is especially true for applications that include support for multiple Android SDK versions, devices, or languages, but it is also important from an update or upgrade perspective. To perform regular, versioned builds, do the following:

- Use a source control system to keep track of project files.
- Version project files at regular intervals and perform routine, reproducible builds.
- Verify (through testing) that each build performs as expected.

There are many wonderful source control systems out there for developers, and most that work well for traditional development will work fine for a mobile project. Many popular

source-control systems—such as Perforce, Subversion, Git, and CVS—work well with Eclipse, including through plug-ins that provide integration right with Eclipse.

Using a Defect Tracking System

A defect tracking system provides a way to organize and track application bugs, or *defects*, and is generally used along with a process for resolving these issues. Resolving a defect generally means fixing the problem and verifying that the fix is correct in a future build.

With mobile applications, defects come in many forms. Some defects occur on all devices, while others occur only on specific devices. Functional defects—that is, features of an application that are not working properly—are only one type of defect. You must look beyond these and test whether an application works well with the rest of the Android operating system in terms of performance, responsiveness, usability, and state management.

Developing Good Test Plans

Testers rely heavily on an application's functional specification, as well as any user interface documentation, to determine whether features and functionality have been properly implemented. The application features and workflow must be thoroughly documented at the screen level and then validated through testing. It is not uncommon for interpretive differences to exist between the functional specification, the developer's implementation, and the tester's resulting experience. These differences must be resolved as part of the defect-resolution process.

Android application testers, or quality-assurance personnel, have a variety of tools at their fingertips. Although some manual testing is essential, there are now numerous opportunities for automated testing to be incorporated into testing plans.

Test plans need to cover a variety of areas, including the following:

- **Functional testing**—This ensures that the features and functions of the application work correctly, as detailed in the application's functional specification.
- **Integration testing**—This ensures that the software integrates well with other core device features on the Amazon Kindle Fire. For example, an application must suspend and resume properly, and it must gracefully handle interruptions from the operating system, such as powering off, modifying the device settings (locking and unlocking the screen), or hitting the Home button.
- **Client/server testing**—Networked mobile applications often have greater testing requirements than standalone applications. This is because you must verify the server-side functionality in addition to the mobile client.
- **Usability testing**—This identifies any areas of the application that lack visual appeal or are difficult to navigate or use, usually from a user-interface perspective. It verifies that the application's resource consumption model matches the target audience for an e-book reader. The Amazon Kindle Fire has very good battery life for reading books, but it's not really well suited to be a hardcore gaming device. Applications should not unnecessarily drain the battery.

- **Performance testing**—This uses the debugging utilities of the Android SDK to monitor memory and resource usage; it also identifies performance bottlenecks and dangerous memory leaks and fixes them.
- **Conformance testing**—This reviews any policies, license agreements, terms, and laws that an application must conform to and verifies that the application complies. For example, the Amazon Appstore has specific content guidelines with which your application must comply. Specifically, the Amazon Appstore prohibits publication of offensive or pornographic content, applications that infringe upon others' copyright intellectual property, or privacy, and allocations that break local, state, national, or international laws.
- **Edge-case testing**—An application must be robust enough to handle random and unexpected events. We've all forgotten to lock our devices on occasion, only to find that the device has received random key presses, launched random apps, or made unnecessary phone calls from the comfort of our pocket. An application must gracefully handle these types of events. That is to say, it shouldn't crash. You can use the monkey tools, Monkey and monkeyrunner, that come with the Android SDK to stress-test an application in both random and reproducible ways.

Maximizing Test Coverage

While 100 percent test coverage is unrealistic, the goal is to test as much of an application as possible, in as many different conditions as possible. To do this, you are likely to need to perform tests on the emulator and on true Amazon Kindle Fire devices, and you may want to consider using both manual and automated testing procedures.

Testing on the Emulator

A test team cannot be expected to set up testing environments on every carrier or in every country where users will use an application. There are times when using the Android emulator can reduce costs and improve testing coverage. The following are some of the benefits of using the emulator:

- Rapidly testing and debugging when physical Amazon Kindle Fire devices are not available or in short supply using AVD emulator settings.
- Simulating devices when they are not yet available (for example, potential preproduction devices in the Amazon Kindle Fire lineup, should you catch wind of any rumors or have a good relationship).
- Testing difficult or dangerous scenarios that are not feasible or recommended on live devices (such as tests that might somehow break a device or invalidate a service agreement).

Testing on Target Devices

Here is a mobile mantra that is worth repeating: *Test early, test often, test on the actual device.*

It's important to get Amazon Kindle Fire devices for testing in-hand as soon as you can. This cannot be said enough: *Testing on the emulator is helpful; testing on the device is essential.* In reality, it doesn't really matter if your application works on the emulator; users run the applications on devices. This also applies to device configurations. For example, it can be convenient to test with the Kindle Fire plugged in, but this is not the way most users will use your application. They will generally only use battery power. Be sure to unplug the Kindle Fire and test an application the way users will most likely encounter it. Pay special attention to how your application affects battery life.

Testing on a target device is the safest way to ensure that an application works correctly, because you are running the application on the same hardware that your users are going to use. By mimicking the environment your users will use, you ensure that your application works as expected in the real world.

Developing applications that specifically target Amazon Kindle Fire devices poses some unique challenges to the developer, because the Kindle Fire is unlike the majority of Android devices in several important ways:

- The Amazon Kindle Fire is not one of the most common device types—it is not a phone, nor is it a generic tablet. It is, at its core, an e-book reader. This means that it does not have many of the commonly used hardware features of Android devices, like sensors, cameras, GPS, or removable storage functionality.
- The Amazon Kindle Fire is not built on the “Google experience,” as many Android devices are. In fact, Google Mobile Services are not available on the Kindle Fire. If your application is also targeting other devices, make sure your design assumptions are compatible with all target devices.
- The Amazon Kindle Fire is not compatible with Android Market features like the In-App billing APIs or the Android License Verification Library (LVL).
- The Amazon Kindle Fire runs Android 2.3.4. It is not likely to receive updates as quickly as other devices. Therefore, it is even more important to ensure that your application uses only compatible SDK features with the appropriate API levels (at this time, API 10 or lower).

Performing Automated Testing

Collecting application information and building automated tests can help you build a better, more bulletproof application. The Android SDK provides a number of packages related to code diagnostics. Application diagnostics fall into these categories:

- Logging application information for performance or usage statistics
- Automated test suites based on the JUnit framework
- Automated testing based on scripts using the monkeyrunner tool

Logging Application Information

At the beginning of this book, you learned how to leverage the built-in logging class `Log` (`android.util.Log`) to implement different levels of diagnostic logging. You can monitor the output of log information from within Eclipse or by using the LogCat utility provided with the Android SDK.

Don't forget to strip any diagnostic information, such as logging information, from the application before publication. Logging information and diagnostics can negatively affect application performance.

Automated Testing with JUnit and Eclipse

The Android SDK includes extensions to the JUnit framework for testing Android applications. Automated testing is accomplished by creating test cases, in Java, that verify that the application works the way you designed it. This automated testing can be done for both unit testing and functional testing, including user-interface testing.

This discussion is not meant to provide full documentation for writing JUnit test cases. For that, look to online resources, such as <http://www.junit.org>, or books on the subject. However, we provide a simple example of how to use JUnit with Android projects in Eclipse.

Automated testing for Android involves just a few straightforward steps:

1. Create a test project.
2. Add test cases to the new project.
3. Run the test project.

The following sections walk you through how to perform each step to test a specific feature of the Have You Read That? settings screen.

Creating the Test Project

Recall from Chapter 1, "Getting Started with Kindle Fire," when you first created a project using Eclipse, that the wizard has an option for creating a test project. You're now going to leverage that option to get quickly up and running with creating test cases. Conveniently, the option for creating a test project is also available after a project already exists. To create a test project for an existing Android project in Eclipse, follow these steps:

1. Select the appropriate project, right-click it, and choose **Android Tools, New Test Project**.
2. On the first wizard dialog (see Figure 16.1, left), enter the name of the project (such as `HYRT_Chapter16_Test`) and pick an appropriate location for it.
3. On the next wizard dialog, labeled **Test Target** (see Figure 16.1, middle), choose **An Existing Android Project** and select the application project to test (for example, `HYRT_Chapter16`).

4. On the final wizard dialog, verify the build target is what you want. The wizard will base this on the test target and, for Kindle Fire, we want it to be API Level 10 (see Figure 16.1, right).

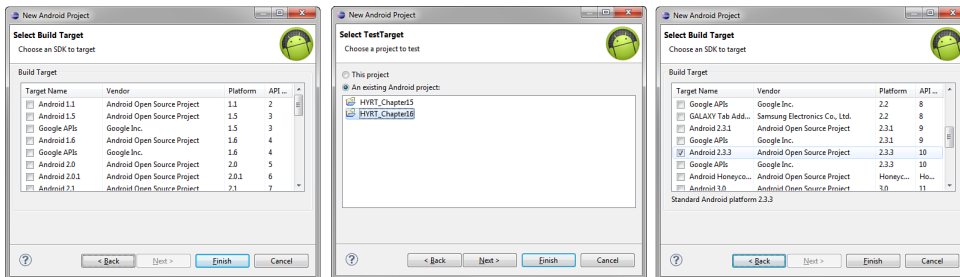


Figure 16.1 Test Application Project Wizard Defaults in Eclipse

5. Click Finish. Your new test project will be created and show up in the Eclipse Package Explorer.

Creating a Test Case

After you have your test project in place, you can write test cases. Let's create a case that tests the behavior of the Nickname field of the settings screen controlled by the `QuizSettingsActivity` class. To do this, first follow these steps to create the empty test-case file:

1. Within your test project, right-click the package name within the `src` folder.
2. Choose New, JUnit Test Case.
3. Set the Name field to `QuizSettingsActivityTests`.
4. Modify the Superclass field to be `android.test.ActivityInstrumentationTestCase2<QuizSettingsActivity>`. (Ignore any warning that says, "Superclass does not exist.")
5. Modify the Class Under Test field to be `com.kindlebook.hyrt.chapter16.QuizSettingsActivity`.
6. Click Finish, as shown in Figure 16.2.
7. In the newly created file, manually add an import statement for `QuizSettingsActivity` (or organize your imports).
8. Add the following constructor to the newly created class:

```
public QuizSettingsActivityTests() {
    super("com.kindlebook.hyrt.chapter16", QuizSettingsActivity.class);
}
```

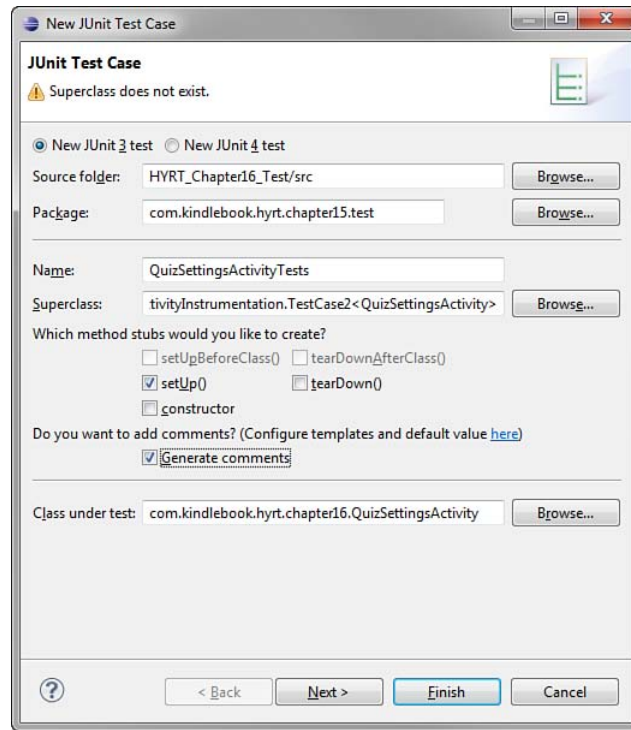


Figure 16.2 Creating a Test Case

Now that your test case file is ready, you can test the Nickname field and make sure that it matches the value of the nickname in `SharedPreferences` and that it updates after a new string is entered. First, modify the `setUp()` method to perform some common behavior. Retrieve the nickname `EditText` object for use in the other two tests. The following code does just that:

```
import com.kindlebook.hyrt.chapter16.R;
...
private EditText nickname;
...
protected void setUp() throws Exception {
    super.setUp();
    final QuizSettingsActivity settingsActivity = getActivity();
    nickname =
        (EditText) settingsActivity.findViewById(R.id.nicknameEntry);
}
```

The import statement is at the file level and must be added manually. The class field is within the test class. The method call for `getActivity()` retrieves the activity being tested. Within an instance of an `ActivityInstrumentationTestCase2` class (our `QuizSettingsActivityTests`, for example), the activity is created as it would normally be when the activity is launched.

Typically, you would also override the `tearDown()` method. However, for these tests, you have no lingering items that need to be cleaned up.

JUnit tests must begin with the word `test`. So, to write specific tests, you need to create methods that begin with the word `test`, followed by what you are testing. First, make sure that the displayed Nickname field is consistent with the stored value in `SharedPreferences`. Add the following code to `QuizSettingsActivityTests` to implement this test:

```
public void testNicknameFieldConsistency() {
    SharedPreferences settings =
        getActivity().getSharedPreferences(QuizActivity.GAME_PREFERENCES,
            Context.MODE_PRIVATE);
    String fromPrefs =
        settings.getString(QuizActivity.GAME_PREFERENCES_NICKNAME, "");
    String fromField = nickname.getText().toString();
    assertTrue("Field should equal prefs value",
        fromPrefs.equals(fromField));
}
```

The first few lines are all standard Android code that you should be familiar with. By using the Android testing framework, you are enabling using the various Android objects within the testing code. The last line, however, is where the real test is performed. The `assertTrue()` method verifies that the second parameter actually is true. If it's not, the string is output in the results. In this case, the two strings are compared. They should be equal.

The next test is to verify that editing the field actually updates the `SharedPreferences` value. Add the following code to `QuizSettingsActivityTests` to test that this is true:

```
private static final String DEBUG_TAG = "QuizSettingsActivityTests";
private static final String TESTNICK_KEY_PRESSES = "T E S T N I C K ENTER";
// ...

public void testUpdateNickname() {
    Log.w(DEBUG_TAG, "Warning: " +
        "If nickname was previously 'testnick' this test is invalid.");
    getActivity().runOnUiThread(new Runnable() {
        public void run() {
            nickname.setText("");
            nickname.requestFocus();
        }
    });
    sendKeys(TESTNICK_KEY_PRESSES);
    ((QuizSettingsActivity)getActivity()).saveFormFields();
    SharedPreferences settings =
```

```

        getActivity().getSharedPreferences(QuizActivity.GAME_PREFERENCES,
            Context.MODE_PRIVATE);
        String fromPrefs =
            settings.getString(QuizActivity.GAME_PREFERENCES_NICKNAME, "");
        assertTrue("Prefs should be testnick", fromPrefs
            .equalsIgnoreCase("testnick"));
    }

```

As before, most of this is standard Android code that you should be familiar with. Not obvious in print, however, is that each letter in the `String TESTNICK_KEY_PRESSES` is separated by a space, except for the command “ENTER”—that is, this is a string of key presses, letters representing the keys corresponding to the letter and “ENTER” representing the Enter key. However, notice that this code is performing a couple calls on the UI thread. This is required for these particular calls; if you remove those calls from the UI thread, the test case fails.

Running Automated Tests

Now that your tests are written, you need to run them to test your code. There are two ways of doing this. The first method is the most straightforward and provides easy-to-read results right in Eclipse: Simply select **Debug, Debug As, Android JUnit Test**. The Console view of Eclipse shows the typical installation progress for both the test application and the application being tested (see Figure 16.3).

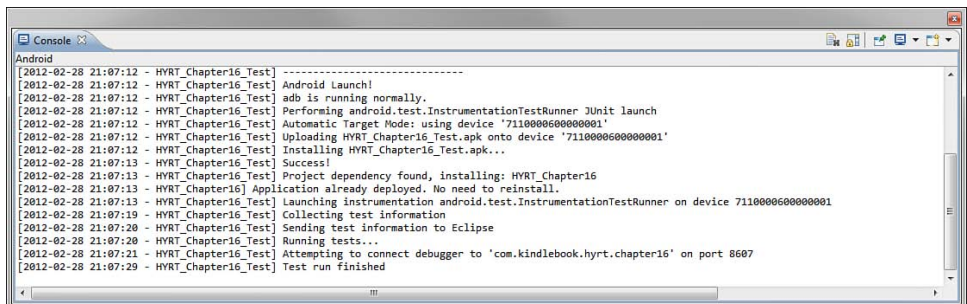


Figure 16.3 Eclipse Console Output While Running JUnit Tests on Android

With the LogCat view, you see the normal Android debug output and new output for the tests that are performed. In this way, you can better debug problems or errors that result from failures or even find new failures that should be tested for.

The JUnit view, however, may be the most useful. It summarizes all the tests run, specifies how long each one takes, and includes a stack trace for any failures found. Figure 16.4 shows what this looks like in Eclipse.

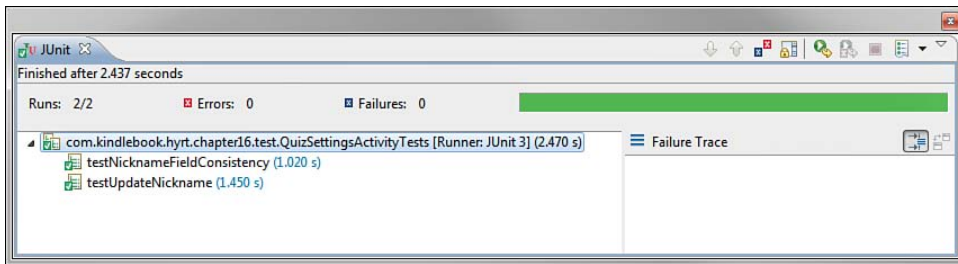


Figure 16.4 Eclipse JUnit View Running Android Tests

The second way of running the tests is available only in the emulator and not on your Kindle Fire. To use this method, launch the Dev Tools app, found installed on the emulator, and then choose Instrumentation. If you've followed along and don't have any other tests installed, you'll likely see `android.test.InstrumentationTestRunner` as the only item shown. Clicking this launches the tests. When you use this method, the only way to see results (other than a visual indication during user interface tests) is to watch the LogCat output.

The description of the item in the list can be changed. In the `AndroidManifest.xml` file of the test app, in the Instrumentation section, modify it to read as follows:

```
<instrumentation
    android:targetPackage="com.kindlebook.hyrt.chapter16"
    android:name="android.test.InstrumentationTestRunner"
    android:label="HYRT Chapter 16 Tests" />
```

Now when you launch Dev Tools and go to the Instrumentation section, it's easier to understand label displays rather than the name.

Adding More Tests

Now you have all the tools you need to add more unit tests to your application. The Android SDK includes a variety of classes that can be implemented for performing a wide range of tests specific to Android. Among these are the following:

- **ActivityUnitTestCase**—Similar to the example testing in the preceding section in that it tests on `Activity`, but at a lower level. This class can be used to unit test specific aspects of an activity, such as how it handles `onPause()`, when it has called `onFinished()`, and so on. This is a great way to test the lifecycle of an activity.
- **ApplicationTestCase**—Like `ActivityUnitTestCase`, this class allows testing of `Application` classes in a fully controlled environment.
- **ProviderTestCase2**—Performs isolated testing on a content provider.
- **ServiceTestCase**—Performs isolated testing on a service.

In addition to these test case objects, there are helper classes for providing mock objects (that is, objects that aren't the real ones but can be used to better trace calls to the real objects), helper classes for simulating touch screen events, and other such utilities. You can find full documentation on these classes in the `android.test` package.

Summary

In this chapter, you learned about the many different ways in which an Android application can be tested and improved, resulting in a higher quality, polished product that Amazon Kindle Fire users will appreciate. You learned many best practices for testing mobile applications, including the importance of creating a solid, complete testing plan. You also learned how to create automated tests using the Android JUnit framework. Finally, you learned about some other specialized testing concerns that should be part of any good product test plan.

Exercises

1. Develop a high-level test plan for the Have You Read That? application.
2. Review the various agreements you have encountered in beginning to develop Android applications (such as the Android SDK License Agreement and the Amazon Appstore developer agreement). Identify any test cases that might be required for compliance with these agreements.
3. Read up on the Monkey Test tools, which are available as part of the Android SDK, at <http://goo.gl/5xJlv>.
4. [Advanced] Know Python? Interested in developing a robust automated testing system for your app? Check out the monkeyrunner tool: <http://goo.gl/ui0B7>.

Registering as an Amazon Application Developer

Your Kindle Fire application may be functionally complete, but you need to take two final steps before you can publish on the Amazon Appstore: You must package the application so that it can be deployed to users safely and securely, and you must sign up for the Amazon Developer Program. In this hour, you learn how to prepare and package an application for release and how to register as an Amazon application developer.

Understanding the Release Process

Preparing and packaging an application for publication is called the *release process*, as shown in Figure 17.1. The release process is an exciting time: The application is stable and working as expected, all those troublesome bugs have been resolved (within reason, at least), and you feel that you're ready to put your app in front of users.

People use different terminology for the release process, and the various software methodologies impose their own terms and processes. Some companies have code names for such milestones, such as “going gold.” Over the years, we've settled on *release* and *release candidate* because, regardless of the methodology of choice, the terms are fairly self-explanatory to most developers.

The final build you perform—the build you expect to deliver to users—is called the release candidate build. The release candidate build should be rigorously tested and verified before it reaches users' hands. If the release candidate build passes every test, it becomes the release build—the official build for publication.

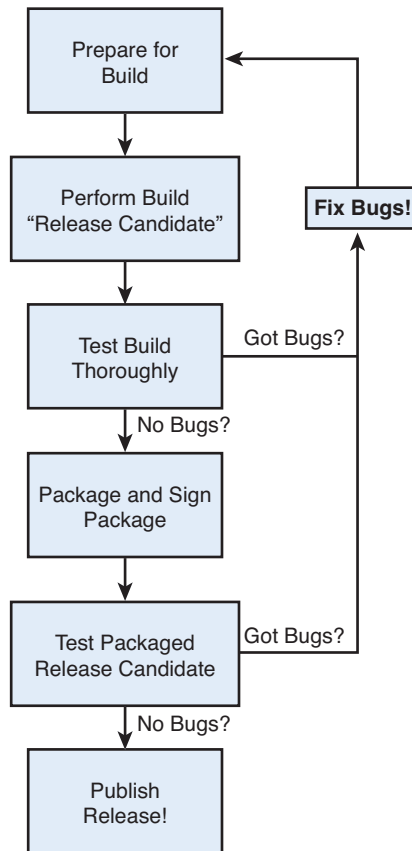


Figure 17.1 An Overview of the Application Release Process

To publish an Android application, follow these steps:

1. Prepare and perform a release candidate build of the application.
2. Thoroughly test the application release candidate.
3. Package and digitally sign the application.
4. Thoroughly test the packaged application release.
5. Publish the application.

Let's explore each step in more detail.

Preparing the Release Candidate Build

It's important to polish your application and make it ready for public consumption. This means that you have to resolve any open or outstanding problems or issues with the application that might block the release. All features must be implemented and tested. All bugs must be resolved or deferred. Finally, you need to remove any unnecessary diagnostic code from the application and verify that the application configuration settings in the Android manifest file are appropriate for release.

Here's a short prerelease checklist for a typical Android application:

- Sufficiently test the application as described in the test plan, including thorough testing on Amazon Kindle Fire under different device configurations (plugged in, battery only, portrait and landscape modes, and so on).
- Fix and verify all defects and bugs in the application.
- Verify that your application meets the guidelines put forth by the Amazon Appstore content guidelines.
- Turn off all debugging diagnostics for release, including any extraneous logging that could affect application performance.

Preparing the Android Manifest File for Release

Before release, you need to make a number of changes to the application configuration settings of the Android manifest file. Some of these changes are simply common sense, and others are required by the Amazon Appstore.

You should review the Android manifest file as follows:

- Verify that the application icon (various sizes of PNG) is set appropriately. This icon will be seen by users and is often used by marketplaces to display the application.
- Verify that the application label is set appropriately. This represents the application name as users will see it.
- Verify that the application version name is set appropriately. The version name is a friendly version label that developers (and marketplaces) use.
- Verify that the application version code is set appropriately. The version code is a number that the Android platform uses to manage application upgrades. Consider incrementing the version code for the release candidate in order to differentiate it from the prerelease version of the application.
- Confirm that the application `<uses-sdk>` setting is set correctly. You can set the minimum, target, and maximum Android SDK versions supported with this build. Recall that Amazon Kindle Fire uses API Level 10.

- Confirm that the application is designed to support the Kindle Fire screen, which is 7" screen (1024×600). Make sure the Manifest file uses the tag for supporting large screens, which is `<supports-screen android:largeScreens="true"/>`.
- Confirm that the application uses only audio formats supported by the Amazon Kindle Fire. The Kindle Fire supports AAC LC/LTP, HE-AACv1 (AAC+), HE-AACv2 (enhanced AAC+), AMR-NB, AMR-WB, MP3, MIDI, Ogg Vorbis, and PCM/WAVE at this time.
- Confirm that the application does not require any features not available on Kindle Fire devices, such as Google Mobile Services (GMS), the Android Market in-app billing APIs, or the License Verification Library (LVL).
- The application should not have any `<uses-feature>` manifest file tags that would cause the application to not be compatible with Kindle Fire devices. For example, your application should not require a gyroscope, camera, WAN module, Bluetooth, microphone, GPS, or micro-SD card to function and should provide alternative behavior if these features are optional.
- Confirm that the application does not violate any of the other guidelines Amazon puts forth for Kindle Fire-compatible applications. For example, applications must not modify themes, act as wallpapers, or customize the lock screen of the device.
- Verify that the `debuggable` option is off.
- Confirm that all application permissions are appropriate. Request only the permissions that the application needs with the `<uses-permission>` tag, and make sure to request permissions the application uses. Note that you cannot customize the lock screen on the Amazon Kindle Fire, so applications cannot include the `android.permission.DISABLE_KEYGUARD` permission.

If required changes are found, don't forget that you need to complete a full testing cycle again to confirm that the changes did not introduce new problems.

Protecting Your Application from Software Pirates

You spent a lot of time, effort, and resources developing your application. The last thing you want is for software pirates to steal your hard work and intellectual property; therefore, you may want to explore the options for obfuscating and protecting your work. Two options available to applications published on the Amazon Appstore are ProGuard obfuscation and Amazon's DRM system.

Protecting Your Application with ProGuard

The Android Eclipse tool-chain includes built-in support for the ProGuard tool to help you secure your application against theft and misuse.

Some of ProGuard's benefits include

- Shrinking and optimizing your application source code. Unused code is removed, making for a leaner package for users to download.

- Obfuscating, or scrambling, your source code, including renaming classes, fields, and methods. This makes “reverse-engineering” your application code more difficult.

Some of ProGuard’s drawbacks include

- Your release build code is obfuscated, making legitimate debugging more of a challenge, but still feasible.
- More advanced ProGuard configurations are needed for applications that use external libraries, paths with spaces, and other coding specifics.

Enabling ProGuard is simple; there is a `proguard.cfg` configuration file associated with your application project in Eclipse where you can modify its settings. Then, you must update the `proguard.config` setting within the application’s `default.properties` configuration file to point at the `proguard` configuration file, like this:

```
proguard.config=proguard.cfg
```

For more information on using ProGuard, see its documentation at the Android developer website (<http://goo.gl/0Lo0G>). We have also written this helpful online article on tips and tricks for using ProGuard to protect your Android applications: <http://goo.gl/cf9UM>.

Protecting Your Application with Amazon’s DRM System

When you submit your application to Amazon Appstore, you can choose to opt in to Amazon’s digital rights management (DRM) system, or you can opt out and have them publish your application DRM free. You cannot use other DRM services with your applications at this time; you must use Amazon’s.

Readying Related Services for Release

If the Android application relies on any external technologies or services, such as an application server, these must also be readied for release.

Many large projects have a “mock” application server (often called a *sandbox*) and a real “live” server. The release build needs to be tested against the live server, just the way users would use it.

Testing the Application Release Candidate

After you address all the prerelease issues, you’re ready to perform the release candidate build. There is nothing particularly special about the general build process, except that you want to launch Run Configuration rather than Debug Configuration in Eclipse.

You should rigorously test the release candidate. In addition to any regular testing, you should verify that the application meets the criteria of the Amazon Appstore in terms of technical and content guidelines.

If you find any defects or issues with the release candidate build, you must decide whether they are serious enough to stop the release process. If you decide that an issue is serious enough to require another build, you simply start the release process over again once you address the issue (see Figure 17.2).

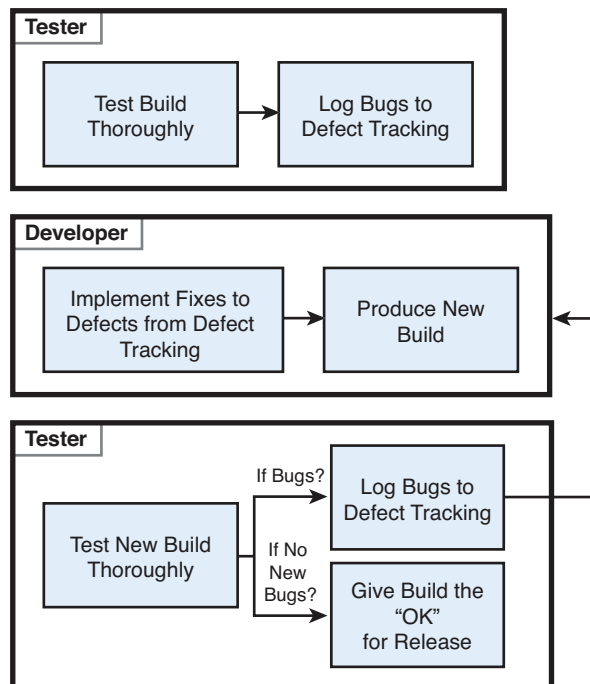


Figure 17.2 The Release Candidate Testing Cycle

Signing Up as an Amazon App Developer

To publish Android applications on Amazon Appstore, you need to join the Amazon Appstore Developer Program. The only way to publish applications for Kindle Fire is through the Amazon Appstore, so this is a requirement for developing for this device.

To sign up, you need a valid Amazon account. Then, go to the Amazon Appstore Developer Program website at <https://developer.amazon.com/welcome.html>, click the Get Started button, and fill in the relevant information for your developer account. We talk more about this process in the next chapter. After you create your account, you can upload your application packages for review.

Packaging and Signing an Application

Now that you have a solid release candidate build that's tested and ready to go, you need to package the application for publication. This process involves generating the Android package file (the `.apk` file) and digitally signing it.

The process of packaging and signing an application has never been easier. Everything you need to package and sign your application is available within the Android plug-in for Eclipse as a simple wizard.

Digitally Signing Applications

Android application packages must be digitally signed for the Android package manager to install them. Throughout the development process, Eclipse has used a debug key to manage this process. However, for release, you need to use a real digital signature—one that is unique to you and your company. To do this, you must generate a private key. A private key identifies the developer and is critical to building trust relationships between developers and users. It is very important to secure private key information.

The private key can be used to digitally sign the release package files of your Android application, as well as any upgrades. This ensures that the application (as a complete entity) is coming from you, the developer, and not someone pretending to be you.

Application updates must be signed with the same private key. For security reasons, the Android package manager does not install the update over the existing application if the key is different. This means you need to keep the key corresponding with the application in a secure, easy-to-find location for future use.

You don't need to use a certificate authority, such as VeriSign or Equifax, that will certify that you are who you say you are before providing a certificate. Self-signing is standard for Android applications, which simply means that you aren't proving who you are, but the next time you publish something, if the keys match, then users (and Android) will know it's been signed by the same person or entity. So, don't share your private key!

Self-Signing and the Amazon Appstore

The Amazon Appstore requires that applications are signed with a certificate. It prefers that you use a certificate provided by Amazon and tied to your Amazon developer account, but you can also self-sign if necessary by obtaining permission from the Amazon Appstore Developer Program by contacting it in writing. After you obtain your certificate, you can sign the application with the method described here.

Exporting and Signing the Package File

You are now ready to export and sign your Android package file. To do this using the wizard provided as part of the Eclipse ADT plug-in, perform the following steps:

1. In Eclipse, right-click the appropriate application project and choose the Export option.
2. Under the Export menu, expand the Android section and choose Export Android Application.
3. Click the Next button.
4. Select the project to export. The one you right-clicked is the default, but you can use the Browse button to change to other open Eclipse projects.
5. Click the Next button.
6. On the keystore selection screen, you can create your own keystore or load the certificate obtained from your Amazon developer account with the associated password. If you choose to create your own, choose the Create New Keystore option and enter a file location (where you want to store the key) as well as a password for managing the keystore. (If you already have a keystore, choose Browse to pick your keystore file, and then enter the correct password.)

Keep Your Keystore Information Secure

Make sure that you choose strong passwords for the keystore. Remember where the keystore is located, too. The same one is required to publish an upgrade to your application. If it's checked in to a revision control system, the password will help protect it, but consider adding an extra layer of privilege required to get to it. Because you will require the key for future upgrades, make sure you have it backed up safely.

7. Click the Next button.
8. On the Key Creation screen, enter the details of the key, including information about your organization. (See the note on key validity.) If you need help with other particular fields, see the Android Developer website documentation on application signing at <http://goo.gl/LWtFj>.
9. Click the Next button.
10. On the Enter Destination and Key/Certificate Checks screen, enter a file destination for the application package file.
11. Click the Finish button.

You have now created a fully signed and certified application package file.

Testing the Signed Application Package

Now that you signed and packaged the application, and now that it's ready for production, you should perform one last test cycle, paying special attention to subtle changes to the installation process for signed applications.

Installing the Signed Application Package

Up until now, you’ve allowed Eclipse to handle the packaging and delivery of the application to handsets and emulators for debugging purposes. Now you have the application release version sitting on your hard drive, and you need to load it and test it.

The simplest way to manually install (or uninstall) an application package (.apk) file on a handset or the emulator is to use the `adb` command-line tool. The following is the command for installing a package using `adb`:

```
adb install <path_to_apk>
```

If there is only one device or emulator, this command works. However, if you have multiple devices and emulators floating around, you need to direct the installation command to a specific one. You can use the `devices` command of the `adb` utility to query for devices connected to your computer:

```
adb devices
```

The list this command returns includes any emulators or handsets attached to the computer. The results might look like this:

```
$ adb devices
List of devices attached
emulator-5554    device
7110000600000001 device
```

You can then target a specific device on which to install the application package file by using the `-s` option. For example, to install the `HaveYouReadThat.apk` application package file on the emulator, use the following:

```
adb -s emulator-5554 install HaveYouReadThat.apk
```

For more information about the `adb` command-line tool, see the website <http://goo.gl/jqXK3>.

Verifying the Signed Application

You’re almost done. Now it is time to perform a few last-minute checks to make sure the application works properly:

- Verify smooth installation of the signed application package.
- Verify that all debugging features have been disabled.
- Verify that the application is using the “live” services as opposed to any “mock” services.
- Verify that application configuration data, such as the application name and icons, as well as the version information, displays correctly.

If you find any issues with the signed application functionality, you must decide whether they are serious enough to stop the release process and begin again. After you thoroughly test the application package and are confident that users will have a positive experience using your application, you are ready to publish!

Summary

In this hour, you learned how to prepare yourself and your application for publication on Amazon Appstore. You learned about the steps to take to verify that your application is ready for publication, such as stripping debugging information, verifying application configuration settings, and performing a release build. You then learned to export an unsigned application package file, generate a private key, and digitally sign the application for publication. Finally, you learned about the signup process for registering as an application developer with Amazon.

Exercises

1. Build and export the Have You Read That? APK package file and digitally sign it (self-signing).
2. Review the package file created in the previous exercise. How large is it? List several ways you might make the package file smaller and leaner. (Hint: Application resources are a big part of the package size.)
3. Install the Have You Read That? application package on an Amazon Kindle Fire (or emulator, if you do not have a device) by using the adb command-line utility.
4. Uninstall the Have You Read That? application package from the device (or emulator, if you do not have a device) by using the Applications section of the device settings.

Publishing Applications on the Amazon Appstore

Congratulations! You've made it to the final chapter, having learned how to design, build, and test an Android application from start to finish. The next logical step is to publish your application. In this chapter, you learn how to publish your Kindle Fire Android applications on the Amazon Appstore. The Android platform supports paid distribution, free distribution, and even self-distribution options. This gives a developer great flexibility for getting applications into the hands of users, with fewer hurdles than most platforms.

Selling on the Amazon Appstore

The Amazon Appstore is a marketplace for Android applications for all Android devices, not just the Amazon Kindle Fire. That said, if you want to target the Amazon Kindle Fire device, you must publish your application through this store by joining the Amazon Appstore Developer Program.

Signing Up for a Developer Account

To publish applications through the Amazon Appstore, you must register as a developer at <https://developer.amazon.com/welcome.html>.

Amazon Appstore Developer Program Fees

Although the Amazon Appstore Developer Program has a fee of \$99 a year, it is currently incentivizing developers by waiving the first year's program fee.

As part of the signup process, you are asked to provide information such as your name, address, phone number, and company information. You then need to review and accept the Developer License Agreement.

Complying with the Developer License Agreement

Read the Developer License Agreement carefully. It is nonexclusive, but it is more restrictive than other publishing options, such as the Android Market. For example, you are required to provide technical and product support to end users and respond to support requests from the Amazon Appstore team in a timely fashion. There are also limitations and guidelines for embedded advertising features often found in free ad-based applications. Furthermore, your application is subject to stringent privacy guidelines. Finally, there are conditions for the timing of application publication and where applications are published at large. These safeguards help ensure a positive end-user experience and make the Amazon Appstore more of a curated application-publishing venue. The idea is that only serious developers will take part.

After you accept the Developer License Agreement, you are made aware of the registration fee (currently \$99, but also waived for the first year as of this writing). Finally, if you plan to distribute paid applications, you can set up your payment information. After you successfully sign up for the Amazon Appstore Developer Program, you can submit your application for review and publication.

Uploading an Application

After you log in to the Amazon Appstore Developer portal account, you can begin publishing applications through the Appstore for Android. From the portal, you can upload your application for review. Be sure to review the content guidelines for Amazon Appstore applications, as well as the details for targeting Amazon Kindle Fire devices with release builds (described in the previous chapter).

You need to fill in numerous details about your application in addition to providing the binary.

Providing Application Details

You need to fill out a number of details for your applications profile information, including various descriptions about the application content. This is where you can opt in or out of the DRM service provided by Amazon. You must also provide various assets for marketing purposes:

- A small application icon (114×114 pixel PNG with transparent background)
- A thumbnail application icon (512×512 pixel PNG with transparent background)
- Several unaltered screenshots of the application in use (3 minimum, PNG or JPG, crop out the taskbar)

You may also want to include

- Promotional graphics for use on the Amazon Appstore home page (290×140 pixels in landscape only or 512×512 pixels, PNG, JPG, GIF [no animation])
- Promotional videos for use on the product detail page (at least 720px wide [4:3 or 16:9], 30 megabytes maximum, 1,200 kbps or higher, formats include MPEG-2, WMV, QuickTime, FLV, AVI, H.264 MPEG-4, and up to 5 videos allowed)

Understanding the Application Approval Process

Once your application is ready to be submitted, you can click the Submit button and start the approval process. When you submit your application, it is given the status of *Submitted*. The application then moves through the review process as the Amazon Appstore developer team reviews and tests your application to ensure that it follows the guidelines set forth in the developer agreement. You will know your application is actively being reviewed when it has the *Under Review* status. Once reviewed, it will either be *Approved*, *Rejected*, or *Pending*. For *Pending* and *Rejected* statuses, you receive an email with an explanation.

After your application successfully makes it through the review process, it is published. Once it's published, the status of the application changes to *Live*. If at some point in the future, the application is removed from the Amazon Appstore, its status is marked as *Suppressed*.

Understanding Amazon Appstore Royalties

At this time, the Amazon Appstore default royalty rate is the greater of 70 percent of the purchase price or 20 percent of the list price. Royalties are paid out approximately one month after the end of the calendar month in which the application sale occurred. For developers in the United States, funds are paid via Electronic Funds Transfer (EFT); for international developers, funds are paid via check. The developer is responsible for managing his own tax situation. There are lower limits on payments: A payment must exceed \$10 for EFT payments and \$100 for check payments, or the payments will be held until royalties owed exceed these values.

Free App of the Day

The Amazon Appstore has a free application download each day. The Amazon Appstore team chooses these applications and makes them available to users at no cost. At the time of this writing, an application that was featured as a free app of the day could reasonably expect to have more than 75,000 downloads that day—a great way to expand your user base quickly. Interested? You can fill out a pitch for your application at the following link: <http://www.amazon.com/gp/html-forms-controller/Amazon-Appstore-Marketing> (<http://goo.gl/SFpPx>).

Using Other Developer Account Benefits

Having a registered Android developer account enables you to manage your applications through the Amazon Appstore Developer Portal. Here, you can designate different user roles to help manage your applications. The following roles are available:

- **Administrator**—The Administrator role has full access to the Developer Portal features. This is generally the account used to set up the developer account in the first place.
- **Analyst**—The Analyst role has limited access to the Developer Portal features, including the ability to access payment and sales reports.

- **Developer**—The Developer role has limited access to the Developer Portal features, including the ability to upload application package files.
- **Marketer**—The Marketer role has limited access to the Developer Portal features, including the ability to manage the company-profile information and the marketing/media materials associated with specific apps. The Marketer role can also access sales reports.

Generating Reports

After you publish your application via the Amazon Appstore Developer Portal, you can generate different reports to see how your application is fairing in the marketplace (provided your account has the appropriate permissions, as just described). Several kinds of reports are available, and you can export the report data via a CSV data file to use in your favorite spreadsheet program or as PDF files.

Sales reports can be used to determine how many users are downloading your application over time. These reports are updated every few hours with new statistics, so you can identify trends in sales. Earnings reports are a monthly feature that can be used to determine your approximate royalty payments per month after sales and refunds have been accounted for. These reports are audited and approved before you review them each month.

Keep in mind that these reports are nonbinding and subject to change. They are meant to give you a ballpark figure of how your application is doing, but not a to-the-penny projection of your sales or earnings. Refunds and reversals, taxes and withholdings, and other details may modify your numbers.

Summary

In this final chapter, you learned how to publish a Kindle Fire application for the world to see and use. Perhaps you already have some great Kindle Fire app ideas in mind. It's time to fire up Eclipse and start coding! When you start building applications, drop us a note and tell us about them. (Our contact information is available in Appendix C, "Supplementary Materials.") We'd love to hear from you!

Exercises

1. Join the Amazon Appstore Developer Program.
2. Browse through the Amazon Appstore (on a Kindle Fire or on the Amazon website). Think of an idea for an application and determine what category and price range is appropriate for that application.
3. Browse through the Amazon Appstore (on a Kindle Fire or on the Amazon website). Focus on two applications—perhaps those similar to your idea or in the same category

(such as Games). Try to find one popular application (high ratings, downloads) and one not-so-popular application. Perform a comprehensive review of these applications' features. Pay special attention to the marketing support materials provided, such as the screenshots, description, and user reviews and ratings. What do they do right? What would you do differently?

4. [Thrilling!] Go write a fabulous and exciting application, and then share it with the world. Email us about your experiences and your app at androidwirelessdev+fireapps@gmail.com.

This page intentionally left blank

IV

Appendixes

- A** Configuring Your Android Development Environment 279
- B** Eclipse IDE Tips and Tricks 289
- C** Supplementary Materials 299

This page intentionally left blank

Configuring Your Android Development Environment

This appendix walks you through the steps needed to install and configure all the appropriate tools to get started developing Android applications for Amazon Kindle Fire devices.

Beware! The Android SDK and Tools Change Frequently

We have made every attempt to provide the latest steps for installing and using the latest tools. However, these steps and the user interfaces described may change at any time without notice. Please review the Android development website (<http://d.android.com/sdk>) and our book website (<http://androidbook.blogspot.com>) for the latest information.

Configuring Your Development Environment

To write Android applications, you must configure your programming environment for development. The Java Development Kit (JDK), the Eclipse development environment, and the Android SDK are available for download on the web at no cost.

To configure your development environment, perform the following steps:

1. Verify that your computer meets the system requirements for Android development.
2. Install a supported version of the JDK.
3. Install a supported development environment, such as Eclipse.
4. Review the Android License Agreement and install the Android SDK Starter Package on your development machine. You can download this package at <http://d.android.com/sdk/>.

5. Install the Android development tools (ADT) Plug-in for Eclipse. This step is completed from within Eclipse as a software update.
6. Use the Android SDK Manager (accessible from within Eclipse using the ADT plug-in) to download and install the appropriate Android platform version (compatible with the Amazon Kindle Fire) and other components.
7. Configure your development computer for USB debugging. In some cases, you may need to install a USB driver in order to properly connect your computer to your Android device.
8. Configure your Amazon Kindle Fire device for development.

Checking for the Most Recent System Requirements

You can find the most recent and accurate list of system requirements for Android development at <http://d.android.com/sdk/requirements.html>.

Development Machine Prerequisites

Android developers may use a number of different operating systems and software configurations. This appendix walks you through the installation of the tools used in this book. If you're installing from scratch, you will want to choose the latest versions of the software packages required for development.

For a complete list of software and system requirements, see the Android developer website at <http://goo.gl/F7i3K>.

Supported Operating Systems

Android applications can be written on the following operating systems:

- Windows XP (32 bit), Vista (32 or 64 bit), or Windows 7 (32 or 64 bit)
- Mac OS X 10.5.8 or later (x86 only)
- Linux (see <http://d.android.com/sdk/requirements.html> for details)

Available Space

You need approximately 2GB of space to safely install all the tools to develop Android applications. This includes installing the JDK, the Eclipse integrated development environment (IDE), the Android SDK, and the tools and plug-ins.

Installing the Java Development Kit

Most Android applications are written in Java, and this book focuses on Android Java development for this reason. Android applications can be developed using Oracle's JDK 5 or JDK 6, which must be installed on the development machine. You can read the license agreement and download the latest version of the Java Standard Edition JDK at Oracle's website (<http://goo.gl/yhhaL>). Make sure that you choose the full JDK for development purposes, not simply the Java Runtime Environment (JRE). Simply follow the directions of the appropriate installer to install the Java development environment on your machine. For specific installation for your operating system, see the documentation available with the installation package you choose.

Installing the Eclipse IDE

This book uses the popular Eclipse IDE for development purposes. Many, if not most, developers use Eclipse for Android development because the Android SDK includes plug-ins that allow tight integration with the IDE. Eclipse is available for the Windows, Mac, and Linux operating systems.

Eclipse comes in various versions. Only versions 3.6 and higher are compatible:

- Eclipse 3.6 (Helios) or
- Eclipse 3.7 (Indigo)

New Versions of Eclipse

Support for Eclipse 3.8 (Juno, due mid 2012) was untested at the time of this writing.

Eclipse comes in multiple packages that include all the tools for Android development. The version you ultimately must pick depends on requirements outside of Android development. The following three are recommended:

- Eclipse Classic
- Eclipse IDE for Java Developers
- Eclipse IDE for Java EE Developers

You can download Eclipse at <http://www.eclipse.org/downloads/>. You can see more about the Eclipse requirements at <http://d.android.com/sdk/requirements.html>.

Installing the Android SDK

Next, you need to install the SDK to develop Android applications. The Android SDK includes the Android JAR file (Android application framework classes) and Android documentation, tools, and sample code. The steps to install the Android SDK vary substantially, depending

on factors such as which operating system you install to and which IDE you use. We highly recommend that you follow the steps provided in the instructions available at the Android developer website (<http://d.android.com/sdk/installing.html>). This website includes information on troubleshooting your installation on various platforms and is updated whenever a new version of the Android SDK or tools is released.

Installing and Configuring the Android Plug-In for Eclipse (ADT)

The Android Plug-in for Eclipse (ADT) enables seamless integration with many of the Android development tools. To install the ADT, you must launch Eclipse and install a custom software update. Software updates are handled in different ways, depending on which version of Eclipse you are using. After installing the plug-in, configure the Android settings within the Eclipse preferences. For step-by-step instructions, see <http://d.android.com/sdk/eclipse-adt.html>.

For example, to install Android Plug-in on Eclipse 3.6 (Helios), follow these steps:

1. Launch Eclipse.
2. Select Help, Install New Software.
3. Click the Add button.
4. Add a repository with the Name ADT and the Location <https://dl-ssl.google.com/android/eclipse/>.
5. Click OK. If this fails to resolve to the appropriate repository, try using [http](http://dl-ssl.google.com/android/eclipse/) in the Location URL instead of [https](https://dl-ssl.google.com/android/eclipse/).
6. You should see items listed in the Available Software listing. Check the checkbox next to Developer Tools to download all available tools.
7. Click the Next button and follow the wizard for installing the tools. Accept the terms of the license agreement and click the Finish button.
8. You may see a warning that you are installing unsigned content. You need to click OK to proceed and install the plug-in.
9. After the software update completes, restart Eclipse as prompted.

Downloading Android SDK Components

The Android tools and SDK versions are componentized. This means that instead of installing one large package for development for all supported versions of Android, you can pick and choose the Android SDK versions you want to install and work with using the Android SDK Manager. This tool allows developers to easily upgrade their development environment when a new version of the Android SDK or tools comes out, which, historically, has happened frequently.

After you install the ADT plug-in, you need to choose and install the specific Android platforms you will develop for, as well as any other components you'd like. For example, Amazon built the Kindle Fire OS using Android 2.3.4, Gingerbread (API Level 10). To install the appropriate components, use the Android SDK Manager as follows:

1. Launch Eclipse.
2. Select Window, Android SDK Manager.
3. Click the Available Packages option on the left-hand menu.
4. You will likely see at least two options: Android Repository and Third Party Add-ons. Most readers will want to download all items, including all sample code, offline documentation, and the tools. However, if you have limited disk space, feel free to limit the components you download to those you require and add others as needed. Select the checkboxes next to the items you want to download. Make sure that you include the Android 2.3.3 (API Level 10) package to target Amazon Kindle Fire devices. Android 2.3.4 is a variant only available on devices with new SDK changes, so it has the same API Level.
5. Click the Install Selected button.
6. Choose the Accept All radio button and click the Install button.
7. You may need to restart components when prompted.
8. When the installation completes, click Close. If you navigate to the Installed Packages menu item, you should see that numerous components and platform versions are now installed.

For this book, we recommend installing most, if not all, of the components available for download, including the following:

- SDK tools and updates (if available, most are installed as part of the Starter Package)
- Android 2.3.3 (API Level 10) package to target Amazon Kindle Fire devices
- Sample applications
- Platform documentation
- USB driver (Windows installations only)

You're better off installing too much than struggling later to determine why something isn't working, only to find that you forgot to install a necessary component. Finally, after you use the Android SDK Manager to download all the Android components for development purposes, update your Eclipse preferences to point at the Android SDK components that you just downloaded and installed. To do this,

1. Launch Eclipse.
2. Select Window, Preferences (or Eclipse, Preferences in Mac OS X).

3. Click the top-level Android preferences and set the SDK location to where you initially installed the Android SDK on your computer.

Upgrading the Android SDK and Tools

The Android SDK and tools are always under development; inevitably, a new version will be released. Use the Android SDK Manager to download the latest and greatest the platform has to offer, but make sure that you continue to use versions that are compatible with Amazon Kindle development. A new version of the Android SDK may involve additions, updates, and removals of libraries; package, class, or method name changes; and updated tools—remember that you want the Android SDK versions to be compatible with the Amazon Kindle Fire.

For each new version of the Android SDK, the following documentation is provided:

- **Platform Highlights**—A brief description of the major changes to the SDK for users and developers
- **API Differences Report**—A complete list of specific changes to the SDK
- **Release Notes**—A list of known issues with the SDK (usually found in the `/docs` directory on your Android SDK installation on your development machine)

These documents are essential reading for an Android developer. In theory, updated SDKs are backward-compatible, although this is not always the case. These documents can provide an important heads-up on new features and help you identify application problems before users' devices receive the upgrade.

Checking for Updates

You will want to routinely check back in the Android SDK Manager to download updates and new versions of the Android SDK tools and components as they become available. Before doing this, make sure to first update the components of Eclipse, which may include plug-in updates. For example, the Android SDK Manager itself is updated through the Eclipse update mechanism.

Debugging with the Amazon Kindle Fire

Much of Android development involves designing applications on your computer and then downloading, running, and debugging them onto Android devices via a USB connection. The following information applies to configuring Amazon Kindle Fire devices for debugging.

The Amazon Kindle Fire does not currently ship with a USB cable, so you need to find one that is compatible. Specifically, you need a USB cable with a Micro-B connection on one end. It's standard fare for Android devices—especially phones—and digital cameras.

Connecting your Amazon Kindle Fire device to your computer requires some manual tweaking, whether you're on a Mac or a Windows machine. You need to get this going before you can start loading apps and debugging directly on the device.

Connecting to a Windows Development Machine

On a Windows development machine, you need to edit two files (creating them if necessary) and then ensure that the connected device is using the correct driver. You can edit these files using your favorite text editor. Because locations vary, we'll reference the file locations with a general description of the location, but you'll have to ultimately find the file for yourself.

First, edit the `adb_usb.ini` file found in the `.android` folder of your user directory. Add one line to the bottom of the file:

```
0x1949
```

Second, edit the `android_winusb.inf` file. You may need to edit the permissions of the file before modifications will be allowed. Add the following two lines (and the comment, if you want) to both the `[Google.NTx86]` and `[Google.NTamd64]` sections:

```
;Kindle Fire
%SingleAdbInterface% = USB_Install, USB\VID_1949&PID_0006
%CompositeAdbInterface% = USB_Install, USB\VID_1949&PID_0006&MI_01
```

Now you're ready to plug in the cable and configure the USB driver. When you connect your Amazon Kindle Fire, your computer will be ready. Once connected, you have to manually choose to use the Google USB driver you downloaded from the Android SDK Manager. You do this from the system's Device Manager, as with any USB device. In brief, from the Device Manager, find the Kindle device, choose Update Driver Software, choose Browse for Driver, find the `android_winusb.inf` file, and install it. Once done, the Kindle device will show up under Android Phones, Android Composite ADB Interface.

Connecting to a Mac Development Machine

On a Mac, the process is slightly different. You only need to edit one file: the `adb_usb.ini` file (creating it if necessary). This file is found in the `.android` folder of your home directory. Add the following two lines to it:

```
0x1949
0x0006
```

After you edit the appropriate files on either Mac or Windows (creating it as an empty file, if needed), be sure to restart `adb` and check to see that the device now appears in the devices list when you perform the `adb devices` command, like so:

```
adb kill-server
adb start-server
adb devices
```

Configuring Other Android Devices for Development Purposes

Although the Amazon Kindle Fire has USB debugging enabled by default and needs no on-device adjustments for debugging, other devices you encounter may. Each Android device may have different debugging settings, but here are the generic steps for enabling Android development settings on an Android device:

1. On the device Home screen, select Menu, Settings, Applications.
2. Click the checkbox to enable Unknown Sources. This allows you to install your applications, as opposed to only applications available on the Android Market.
3. Select the Development menu (Menu, Settings, Applications, Development).
4. Click the checkbox to enable USB debugging. This allows you to debug your applications while they are running on this device from within Eclipse.
5. Optionally, click the checkbox to enable Stay Awake. This keeps the device from going to sleep during long debugging sessions.

Configuring Your Operating System for Device Debugging

To install and debug Android applications, you might need to configure your operating-system drivers so that you can connect to devices via USB. This is especially true of Windows machines. The Android SDK ships with drivers compatible with most Android devices, including Amazon Kindle Fire devices.

Notes on Windows Installations

If you're running on a Windows operating system, you will likely need to install Android USB drivers. These can be downloaded from the Available Packages section of the Android SDK Manager. Once downloaded, you can use the Device Manager and point at the `google-usb_driver` folder under the Android SDK directory. Alternatively, you can download the latest Google USB drivers from the Android website at <http://goo.gl/TkqjL> and get a list of sources to find specific manufacturer USB drivers at <http://goo.gl/ecNHn>. After you unzip the drivers, connect your phone to your computer via the USB cable and select the drivers you want to install.

Notes on Mac OS X Installations

On a supported Mac, all you have to do is plug in the USB cable to the Mac and the device. No additional configuration is needed.

Problems with the Android SDK

You might occasionally come across problems with the SDK. If you think you found a problem, you can find a list of open issues and their statuses at the Android project Issue Tracker website (<http://code.google.com/p/android/issues/list>).

You can report bugs for review by the Android team at <http://source.android.com/source/report-bugs.html>. Always check if there is an existing open bug before creating a new one. If there is, add your information to that defect. When creating a new defect, include as much information as possible when reporting a potential bug. (Sample code often helps.)

This page intentionally left blank

Eclipse IDE Tips and Tricks

The Eclipse integrated development environment (IDE) is the most popular development environment for Android developers. In this appendix, we provide a number of helpful tips and tricks for using Eclipse to develop Android applications quickly and effectively.

Share Your Own Eclipse Tricks with Us!

Do you have your own tips or tricks for Android development in Eclipse? If so, email them to us (with permission to publish them) at androidwirelessdev@gmail.com, and they might be included on our blog at <http://androidbook.blogspot.com> or in the next edition of one of our books. Get your moment of geekly fame!

Organizing Your Eclipse Workspace

In this section, we provide a number of tips and tricks to help you organize your Eclipse workspace for optimum Android development.

Integrating with Source Control Services

Eclipse has the ability to integrate with many source-control packages using add-ons or plugins. This allows Eclipse to manage checking out a file (making it writable) when you first start to edit a file, checking a file in, updating a file, showing a file's status, and numerous other tasks, depending on the support of the add-on.

Eclipse Works with Many Source Control Systems

Common source-control add-ons are available for CVS, Subversion, Perforce, git, Mercurial, and many other packages.

Generally speaking, not all files are suitable for source control. For Android projects, any file within the `/bin` and `/gen` directories shouldn't be in source control. To exclude these generically within Eclipse, go to Preferences, Team, Ignored Resources. You can add file suffixes such as `*.apk`, `*.ap_`, and `*.dex` by clicking the Add Pattern button and adding one at a time. Conveniently, this applies to all integrated source-control systems.

Repositioning Tabs Within Perspectives

Eclipse provides some pretty decent layouts with the default perspectives. However, not every one works the same way. We feel that some of the perspectives have poor default layouts for Android development and could use some improvement.

Customize Eclipse for Yourself

Experiment to find a tab layout that works well for you. Each perspective has its own layout, too, and the perspectives can be task-oriented.

For example, the Properties tab is usually found on the bottom of a perspective. For code, this works fine, because this tab is only a few lines high. But, for resource editing in Android, this doesn't work so well. Luckily, in Eclipse, this is easy to fix: Simply drag the tab by left-clicking and holding on the tab (the title) itself and dragging it to a new location, such as the vertical section on the right side of the Eclipse window. This provides the much needed vertical space to see the dozens of properties often found here.

Resetting Perspectives

If you mess up a perspective or just want to start fresh, you can reset it by choosing Window, Reset Perspective.

Maximizing Windows

Sometimes, you might find that the editor window is just too small, especially with all the extra little metadata windows and tabs surrounding it. Try this: Double-click the tab of the source file that you want to edit. Boom! It's now nearly the full Eclipse window size! Just double-click to return it to normal. (Ctrl+M works on Windows, and Cmd+M works on the Mac.)

Minimizing Windows

You can minimize entire sections, too. For example, if you don't need the section at the bottom that usually has the console or the one to the left that usually has the File Explorer view, you can use the minimize button in each section's upper-right corner. Use the button that looks like two little windows to restore it.

Viewing Windows Side-by-Side

Ever wish that you could see two source files at once? Well, you can! Simply grab the tab for a source file and drag it either to the edge of the editor area or to the bottom. You then see a dark outline, showing where the file will be docked—either side-by-side with another file or above or below another file. This creates a parallel editor area where you can drag other file tabs as well. You can repeat this multiple times to show three, four, or more files at once.

Tabs and Windows Outside Eclipse Frame

Drag a tab or a window outside the Eclipse frame, and it will create a new system-level window with the tab or editor window in it. You can also create an entirely new system-level window containing a full perspective view with Window, New Window. Either of these methods is particularly useful for taking advantage of multiple monitors with Eclipse.

Viewing Two Sections of the Same File

Ever wish that you could see two places at once in the same source file? You can! Right-click the tab for the file in question and choose New Editor. A second editor tab for the same file comes up. With the previous tip, you can now have two different views of the same file.

Closing Unwanted Tabs

Ever feel like you get far too many tabs open for files that you're no longer editing? I do! There are a number of solutions to this problem. First, you can right-click a file tab and choose Close Others to close all other open files. You can quickly close specific tabs by middle-clicking each tab. (This even works on a Mac with a mouse that can middle-click, such as one with a scroll wheel.)

Keeping Windows Under Control

Finally, you can use the Eclipse setting that limits the number of open file editors:

1. Open Eclipse's Preferences dialog.
2. Expand General, choose Editors, and check Close Editors Automatically.
3. Edit the value in Number of Opened Editors Before Closing.

This causes old editor windows to be closed when new ones are open. Eight seems to be a good number to use for the Number of Opened Editors Before Closing option to keep the clutter down but to have enough editors open to still get work done and have reference code open. Note also that if you check Open New Editor under When All Editors Are Dirty or Pinned, more files will be open if you're actively editing more than the number chosen. Thus, this setting doesn't affect productivity when you're editing a large number of files all at once, but it can keep things clean during most normal tasks.

Creating Custom Log Filters

Every Android log statement includes a tag. You can use these tags with filters defined in LogCat. To add a new filter, click the green plus sign button in the LogCat pane. Name the filter—perhaps using the tag name—and fill in the tag you want to use. Now, there is another tab in LogCat that shows messages that contain this tag. In addition, you can create filters that display items by severity level.

Android convention has largely settled on creating tags based on the name of the class. You see this frequently in the code provided with this book. Note that we create a constant in each class with the same variable name to simplify each logging call. Here's an example:

```
public static final String DEBUG_TAG = "MyClassName";
```

This convention isn't a requirement, however. You could organize tags around specific tasks that span many activities, or you could use any other logical organization that works for your needs. Another simpler way to do this is as follows:

```
private final String DEBUG_TAG = getClass().getSimpleName();
```

Although not as efficient at runtime, this code can help you avoid copy-and-paste errors. If you've ever been looking over a log file and had a misnamed debug tag string mislead you, this trick may be useful to you.

Searching Your Project

You have several ways to easily search your project files from within Eclipse. The search options are found under the Search menu of the Eclipse toolbar. Most frequently, we use the File Search option, which allows you to search for text within the files found in the workspace, as well as files by name. The Java Search option can also help you find Java-specific elements of your project, such as methods and fields.

Organizing Eclipse Tasks

By default, any comment that starts with `// TODO` shows up on the Tasks tab in the Java perspective. This can be helpful for tagging code areas that require further implementation. You can click a specific task, and it will take you straight to the comment in the file so you can implement the item at a later time.

You can also create custom comment tags above and beyond to-do items. We often leave comments with a person's initials to make it easy for them to find specific functional areas of the application to code review. Here's an example:

```
// LED: Does this look right to you?
// CDC: Related to Bug 1234. Can you fix this?
// SAC: This will have to be incremented for the next build
```

You might also use a special comment, such as `//HACK:`, when you have to implement something that is less than an ideal implementation, to flag that code as subject to further review.

To add custom tags to your Task list, edit your Eclipse preferences (available at Window, Preferences or Eclipse, Preferences on a Mac) and navigate to Java, Compiler, Task Tags. Add any tags that you want to flag. The tags can be flagged for a certain priority level. So, for example, something with your initials might be high priority to look at right away, but a HACK flag may be low priority because, presumably, it works but maybe not in the best possible way.

Writing Code in Java

In this section, we provide a number of tips and tricks to help you implement the code for your Android applications.

Using Autocomplete

Autocomplete is a great feature that speeds up code entry. If this feature hasn't appeared for you yet or has gone away, you can bring it up by pressing Ctrl+spacebar. Autocomplete not only saves you time typing, but can be used to jog your memory about methods—or help you find a new method. You can scroll through all the methods of a class and even see the Javadocs associated with them. You can easily find static methods by using the class name or the instance variable name. You follow the class or variable name with a dot (and maybe Ctrl+spacebar) and then scroll through all the names. Then, you can start typing the first part of a name to filter the results.

Creating New Classes and Methods

You can quickly create a new class and corresponding source file by right-clicking the package to create it and then choosing New, Class. Next, you enter the class name, pick a superclass and interfaces, and choose whether to create default comments and method stubs for the superclass for constructors or abstract methods.

Along the same lines as creating new classes, you can quickly create method stubs by right-clicking a class or within a class in the editor and choosing Source, Override/Implement Methods. Then, you choose the methods for which you're creating stubs, where to create the stubs, and whether to generate default comment blocks.

Organizing Imports

When referencing a class in your code for the first time, you can hover over the newly used class name and choose Import Classname (package name) to have Eclipse quickly add the proper import statement.

In addition, the Organize Imports command (Ctrl+Shift+O in Windows and Cmd+Shift+O on a Mac) causes Eclipse to automatically organize your imports. Eclipse removes unused imports and adds new ones for packages used but not already imported.

If there is any ambiguity in the name of a class during automatic import, such as with the Android Log class, Eclipse prompts you with the package to import. Finally, you can configure Eclipse to automatically organize the imports each time you save a file. This can be set for the entire workspace or for an individual project.

Configuring this for an individual project gives you better flexibility when you're working on multiple projects and don't want to make changes to some code, even if the changes are an improvement. To configure this, perform the following steps:

1. Right-click the project and choose Properties.
2. Expand Java Editor and choose Save Actions.
3. Check Enable Project Specific Settings, Perform the Selected Actions on Save, and Organize Imports.

Formatting Code

Eclipse has a built-in mechanism for formatting Java code. Formatting code with a tool is useful for keeping the style consistent, applying a new style to old code, or matching styles with a different client or target (such as a book or an article).

To quickly format a small block of code, select the code and press Ctrl+Shift+F in Windows (or Cmd+Shift+F on a Mac). The code is formatted to the current settings. If no code is selected, the entire file is formatted. Occasionally, you need to select more code—such as an entire method—to get the indentation levels and brace matching correct.

The Eclipse formatting settings are found in the Properties pane under Java Code Style, Formatter. You can configure these settings on a per-project or workspace-wide basis. You can apply and modify dozens of rules to suit your own style.

Renaming Almost Anything

Eclipse's Rename tool is quite powerful. You can use it to rename variables, methods, class names, and more. Most often, you can simply right-click the item you want to rename and choose Refactor, Rename. Alternatively, after selecting the item, you can press Ctrl+Alt+R in Windows (or Cmd+Alt+R on a Mac) to begin the renaming process. If you are renaming a top-level class in a file, the filename has to be changed as well. Eclipse usually handles the source control changes required to do this, if the file is being tracked by source control. If Eclipse can determine that the item is in reference to the identically named item being renamed, all instances of the name are also renamed. Occasionally, this even means that comments are updated with the new name. Handy!

Refactoring Code

Do you find yourself writing a whole bunch of repeating sections of code that look, for example, like the following?

```

TextView nameCol = new TextView(this);
nameCol.setTextColor(getResources().getColor(R.color.title_color));
nameCol.setTextSize(getResources().getDimension(R.dimen.help_text_size));
nameCol.setText(scoreUserName);
table.addView(nameCol);

```

This code sets text color, text size, and text content. If you've written two or more blocks that look like this, your code could benefit from refactoring. Eclipse provides two useful tools—Extract Local Variable and Extract Method—to speed this task and make it almost trivial.

Using the Extract Local Variable Tool

Follow these steps to use the Extract Local Variable tool:

1. Select the expression `getResources().getColor(R.color.title_color)`.
2. Right-click and choose Refactor, Extract Local Variable (or press Ctrl+Alt+L).
3. In the dialog that appears, enter a name for the variable and leave the Replace All Occurrences checkbox selected. Then, click OK and watch the magic happen.
4. Repeat steps 1–3 for the text size.

The result should now look like this:

```

int textColor = getResources().getColor(R.color.title_color);
float textSize = getResources().getDimension(R.dimen.help_text_size);
TextView nameCol = new TextView(this);
nameCol.setTextColor(textColor);
nameCol.setTextSize(textSize);
nameCol.setText(scoreUserName);
table.addView(nameCol);

```

All repeated sections of the last five lines also have this change made. How convenient is that?

Using the Extract Method Tool

Now, you're ready for the second tool. Follow these steps to use the Extract Method tool:

1. Select all five lines of the first block of code.
2. Right-click and choose Refactor, Extract Method (or choose Ctrl+Alt+M).
3. Name the method and edit the variable names to anything you want. (Move them up or down, too, if desired.) Then, click OK and watch the magic happen.

By default, the new method is below your current one. If any other blocks of code are actually identical (meaning the statements of the other blocks must be in the exact same order), the types are all the same, and so on, they will also be replaced with calls to this new method! You can see this in the count of additional occurrences shown in the dialog for the Extract Method

tool. If that count doesn't match what you expect, check that the code follows exactly the same pattern. Now, you have code that looks like the following:

```
addTextToRowWithValues(newRow, scoreUserName, textColor, textSize);
```

It is easier to work with this code than with the original code, and it was created with almost no typing! If you had ten instances before refactoring, you saved a lot of time by using this useful Eclipse feature.

Reorganizing Code

Sometimes, formatting code isn't enough to make it clean and readable. Over the course of developing a complex activity, you might end up with a number of embedded classes and methods strewn about the file. A quick Eclipse trick comes to the rescue.

With the file in question open, make sure that the outline view is also visible. Simply click and drag methods and classes around in the outline view to place them in a suitable logical order. Do you have a method that is only called from a certain class but available to all? Just drag it into that class. This works with almost anything listed in the outline, including classes, methods, and variables.

Using QuickFix

The QuickFix feature, which is accessible under Edit, QuickFix (or Ctrl+1 in Windows and Cmd+1 on a Mac), isn't just for fixing possible issues. It brings up a menu of various tasks that can be performed on the highlighted code, and it shows what the change will look like. One useful QuickFix now available is the Android "Extract String" command. Use QuickFix on a string literal, and you can quickly move it into an Android strings resource file, and the code is automatically updated to use the string resource. Consider how QuickFix Extract String would work on the following two lines:

```
Log.v(DEBUG_TAG, "Something happened");
String otherString = "This is a string literal.";
```

The updated Java code is shown here:

```
Log.v(DEBUG_TAG, getString(R.string.something_happened));
String otherString = getString(R.string.string_literal);
```

And these entries have been added to the string resource file:

```
<string name="something_happened">Something happened</string>
<string name="string_literal">This is a string literal.</string>
```

The process also brings up a dialog for customizing the string name and which alternative resource file it should appear in, if any.

The QuickFix feature can be used in layout files with many Android-specific options for performing tasks such as extracting styles, extracting pieces to an include file, wrapping pieces in a new container, and even changing the widget type.

Providing Javadoc-Style Documentation

Regular code comments are useful (when done right). Comments in Javadoc style appear in code-completion dialogs and other places, thus making them even more useful. To quickly add a Javadoc comment to a method or class, simply press Ctrl+Shift+J in Windows (or Cmd+Alt+J on a Mac). Alternatively, you can choose Source, Generate Element Comment to prefill certain fields in the Javadoc, such as parameter names and author, thus speeding the creation of this style of comment. Finally, if you simply start the comment block with `/**` and press Enter, the appropriate code block will be generated and prefilled as before.

Resolving Mysterious Build Errors

Occasionally, you might find that Eclipse is finding build errors where there were none just moments before. In such a situation, you can try a couple quick Eclipse tricks.

First, try refreshing the project: Simply right-click the project and choose Refresh (or press F5). If this doesn't work, try deleting the `R.java` file, which you can find under the `gen` directory under the name of the particular package being compiled. (Don't worry: This file is created during every compile.) If the Compile Automatically option is enabled, the file is re-created. Otherwise, you need to compile the project again.

A second method for resolving certain build errors involves source control. If the project is managed by Eclipse via the Team, Share Project menu selection, Eclipse can manage files that are to be read-only or automatically generated. Alternatively, if you can't or don't want to use source control, make sure that all the files in the project are writeable (that is, not read-only).

Finally, you can try cleaning the project. To do this, choose Project, Clean, and choose the project(s) that you want to clean. Eclipse removes all the temporary files and then rebuilds the project(s). If the project was an NDK project, don't forget to recompile the native code.

This page intentionally left blank

Supplementary Materials

This book introduces Android, but this short “crash course” barely scratches the surface of the platform. This book is meant to be used along with supplementary book materials, including the accompanying source code, the publisher’s website, the authors’ book website, and the up-to-date documentation provided with the Android SDK. A number of supplementary materials have been developed especially for this book. These materials, such as source code for many of the examples provided, are available online. There are also a number of other online resources available for Android developers.

Using the Source Code for This Book

The source code for this book is designed with the assumption that you’ll follow along with the accompanying chapter text. The source code downloads are not the “answers” to the lessons and exercises. Because of length restrictions, we are unable to provide pages and pages of code listings in this book. Instead, we provide code snippets on the topics at hand and expect the reader to see the source code if they need further clarification. The source code for this book is available in two locations: on the publisher’s website and on the authors’ book website.

We make every effort to make the code in this book both forward- and backward-compatible, but we have no control over the changes made by the Android team.

The source code for this book functioned as designed when this book was published and was tested with the exact versions of the tools and Android SDK referenced in this book’s Introduction. However, subsequent Android SDK and tool releases sometimes introduce changes; therefore, you may want to check for the latest version of the source code or the authors’ website if you run into problems.

The source code is especially helpful for

- Understanding the full scope of a feature’s implementation, beyond what is discussed in the code excerpt in the book text

- Clarifying Java implementation details for those with limited (or rusty) Java experience
- Providing a fully functional implementation of the concepts for a given lesson
- Providing hints or even implementations of material from the exercises

Accessing the Android Developer Website

Just as you wouldn't get very far learning a foreign language without a textbook and a dictionary for translation, it is impossible to master Android without using the SDK class documentation. The Android Developer website and SDK documentation is available at <http://developer.android.com>, as discussed in Chapter 2, "Mastering the Android Development Tools." The Android Developer website is especially helpful for

- Researching Android SDK APIs, classes, and methods used in this book
- Finding additional tutorials or articles on topics discussed in this book
- Keeping up with the latest trends and revisions of the Android SDK
- Diving deeper into a topic not covered in detail in this introductory book

Accessing the Publisher's Website

The source code that accompanies this book is available for download from the publisher's website (www.informit.com/title/9780321833976) (see Figure C.1).

Here's what you'll find on the publisher's website:

- A thorough description of this book
- Downloadable source code
- Errata and book updates
- InformIT users' reviews of this book
- Sample content
- Other related books

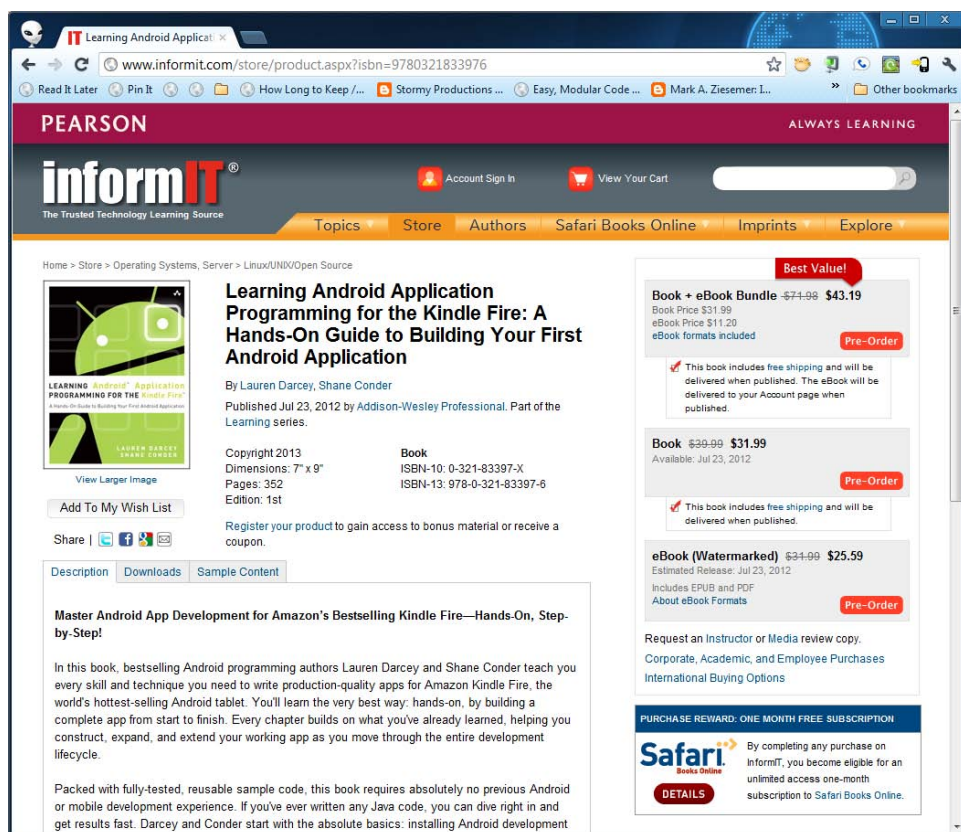


Figure C.1 The InformIT Website

Accessing the Authors' Website

The authors' book website (<http://androidbook.blogspot.com>) is a complementary guide for designing, developing, debugging, and distributing Android applications (see Figure C.2); the source code is also available for download here.

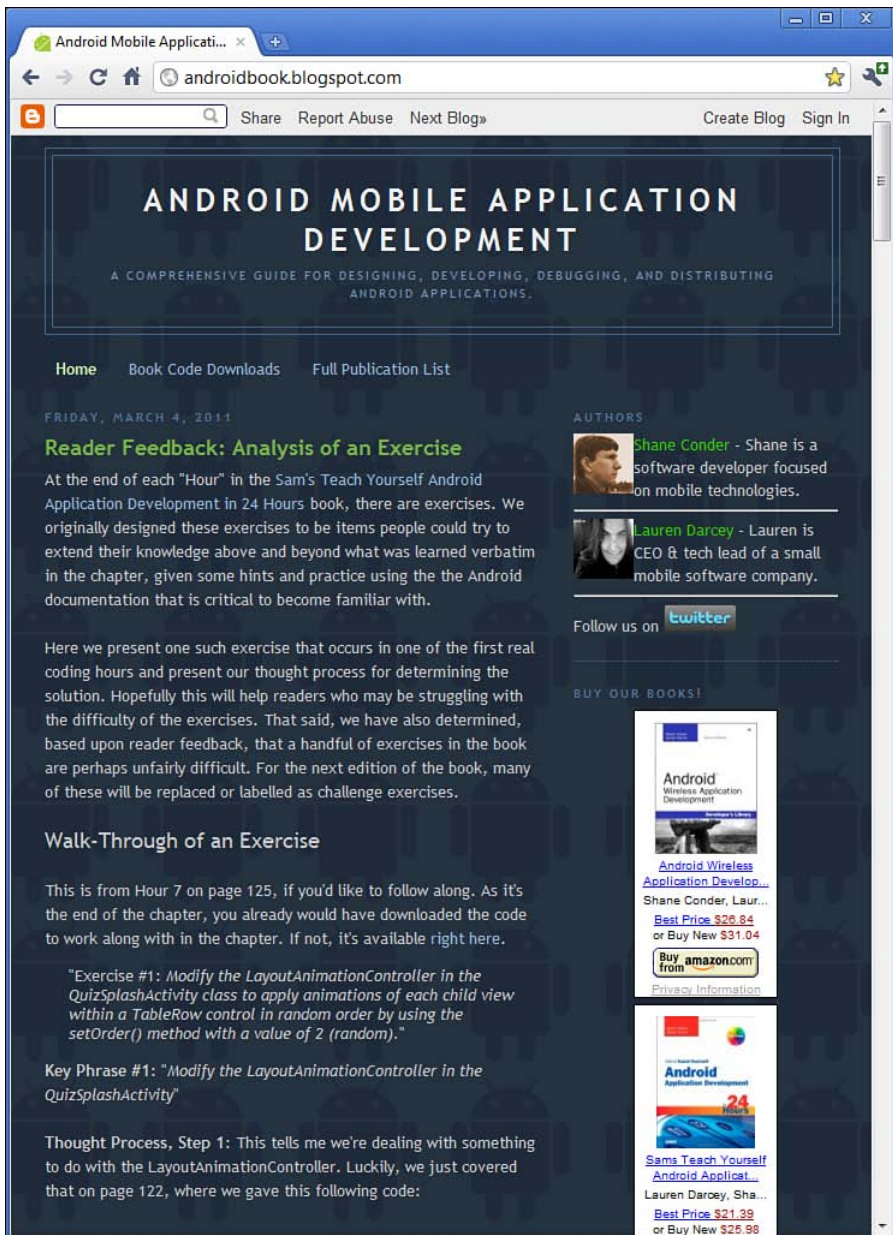


Figure C.2 The Authors' Website

Here's what you'll find on the authors' website:

- Downloadable source code
- Clarification regarding book exercises and reader questions
- Information about Android SDK updates and revisions, especially if a change affects readers
- More technical articles for Android developers written by the authors
- Market news and information related to Android and mobile
- Tips, tricks, and pitfalls of Android development
- Links to reviews of the authors' books
- Supplemental code examples
- Informal discussions of more advanced Android development topics
- Links to other Android materials written by the authors, including their more advanced Android book and technical articles, many of which are available online

Contacting the Authors

We do our best to answer each and every question, and we often post commonly asked questions and their answers on this book's website (<http://androidbook.blogspot.com>).

As always, we welcome your feedback! If you have comments, questions, or concerns about the content of this book, email us (Lauren and Shane) at androidwirelessdev+kf1@gmail.com (see Figure C.3).



Figure C.3 Send Us Feedback!

Leveraging Online Android Resources

The Android developer community is friendly and helpful. Here are a number of useful websites for Android developers and followers of the wireless industry in general:

- **Android Developer website**—The Android SDK, developer reference site, and forums (<http://developer.android.com>)
- **Amazon Appstore Developer Portal**—Resources for Android developers targeting the Amazon Appstore and Kindle Fire devices (<https://developer.amazon.com/home.html>)
- **Stack Overflow: Android**—A collaborative site for programmers, with an official section for Android (<http://stackoverflow.com/questions/tagged/android>)
- **Wireless Developer Network**—A daily news digest for the wireless industry (<http://www.wirelessdevnet.com>)
- **Developer.com**—A developer-oriented site that publishes technical articles (<http://www.developer.com>)
- **Open Handset Alliance**—Android manufacturers, operators, and developers (<http://www.openhandsetalliance.com>)
- **OpenIntents**—An Android developer resource with a public intent registry and a source for third-party Android libraries and extensions (<http://openintents.org>)
- **anddev.org**—An Android developer forum (<http://www.anddev.org>)
- **Google Plus Android Developers**—The Android developer team at Google, on Google Plus (<https://plus.google.com/108967384991768947849/posts> [<http://goo.gl/FWYtN>])
- **FierceDeveloper**—A weekly newsletter for wireless developers (<http://www.fiercedev.com>)

Index

A

accessing

- Amazon Web Services SDK, 226
- Android Developer websites, 300
- applications
 - functionality, 44
 - preferences, 42-43
- author's website, 301-303
- communication protocols, 199
- networks
 - resources, 80
 - services, 202-204
- permissions, 79-81, 203
- publisher's websites, 300
- raw files, 64
- raw resource files, 137
- resources, 53
- XML, 64

accounts

- developers
 - benefits of, 273-274
 - registering, 271
- registering, 266

activities

- applications, implementing, 92-93
- applying, 43-45
- defining, 77-79
- launching, 43-44, 78-79
- networks, 201
- options menus, adding, 130
- ProgressBar control, 204-205
- registering, 77-78
- shutting down, 45
- state, managing, 44-45
- trivia games, 84-85

Activity class, 15, 40-41

- DatePickerDialog class, 168-171
- dialogs, 165-168
 - customizing, 171-177
 - types of, 166-167

- layout files, 40
- methods, 48

adding

- DatePickerDialog to classes, 168-169
- dialogs, customizing, 172-175
- drawable resources, 92
- existing projects, 15
- GridView control, 122-123
- headers, scores screens, 147
- locales, 241
- logic to applications, 181
- network support, 199
- options menus, 129-130
- resources, 91-92, 135
 - colors, 153
 - main menu screens, 120-121
 - raw files, 136
 - splash screens, 103-106
 - strings, 153
- tabbing support, 139
- TabHost control, 139
- tabs, 145
- testing, 259
- XML files, 58

addTab() method, 145

- ADTs (Android Development Tools), 12, 280
 - installing, 282

advanced graphics animation, 110

alarms, 44

AlertDialog, 166, 173

alternative hierarchies, resources, 233-236

- alternative resources. *See also* resources
 - programming, 236
 - qualifiers, managing, 234-236
 - rules, 234

Amazon

- Appstore, 244
- Developer Portal, 304
- developers, registering as, 266
- publishing, 271. *See also* publishing
- reports, 274

- royalties, 273
- selling on, 271-274
- CloudWatch, 228
- DynamoDB, 227
- Elastic Load Balancing, 228
- S3, 227
- Simple Email Service (SES), 227
- Simple Notification Service (SNS), 227
- Simple Queue Service (SQS), 228
- Web Services SDK, 225
 - navigating, 226-228
 - overview of, 225-226
- anddev.org, 304**
- AndroidManifest.xml tab, 73**
- Android**
 - Developer websites, 300, 304
 - Development Tools. *See* ADTs
 - SDK
 - downloading, 282-284
 - installing, 281-282
 - Manager, 283, 284
 - troubleshooting, 287
 - upgrading, 284
 - Virtual Device. *See* AVD
- AndroidManifest.xml tab, 18**
- android.permission.INTERNET permission, 190**
- android.permission.WAKE_LOCK permission, 203**
- android.util.log class, 50**
- animation**
 - applying, 110-115
 - image settings, 113-114
 - sequences, 66
 - splash screens, 101
 - design, 101
 - implementing, 102-109
 - views, 112
- APIs (application programming interfaces), 75-76, 228**
- App Engine (Google), 200**
- Apple iPhone, 9**
- Application Manager, 96**
- Application Nodes section, 78**
- application programming interfaces. *See* APIs**
- <application> tag, 76**
- Application tab, 71**
 - Application Nodes section, 78
- applications**
 - activities, implementing, 92-93
 - attributes, 77
 - builds, 20-28, 39
 - activities, 43-45
 - context, 42-43
 - dialog boxes, 48-49
 - fragments, 49-50
 - intents, 46-48
 - logging, 50-51
 - tracking, 74
 - customizing, 237-240
 - DDMS, debugging, 31-35
 - deployment, 12
 - descriptions, 76
 - design, 39-41
 - development, 11
 - devices, launching, 25-28
 - DRM, 265
 - environments, configuring, 279-280
 - frameworks
 - design, 83
 - trivia games, 83-89
 - functionality
 - accessing, 44
 - implementing, 41
 - Have You Read That?, 84. *See also* Have You Read That? application
 - icons, 76
 - internationalizing, 240-241, 245-246
 - JDK, installing, 281
 - launching, 22-25, 47
 - locales, 241-243
 - localization, 241-243
 - logic
 - adding, 181
 - implementing, 190-196
 - manifest files, 18, 73-77
 - naming, 15, 76
 - networks
 - design, 199-201
 - developing, 201-202
 - packages, 12, 267-270
 - permissions, managing, 79-81
 - piracy, 264-265
 - preferences
 - accessing, 42-43
 - configuring, 93-95

- prerelease checklists, 263
- projects
 - adding resources, 91-92
 - creating new, 90-91
- prototypes
 - implementing, 90-95
 - running, 95-97
- publishing, 271
- registering, 261
- release processes, 261-262
- resources
 - applying, 53-56
 - managing, 53, 236
 - references, 55-56
 - retrieving, 42
 - storing, 54
 - values, 57-59
- servers, applying, 199-200
- settings, 81
- signing, 267-268
- targets, deploying, 27
- testing, 249
 - best practices, 249-252
 - maximizing coverage, 252-260
- uploading, 272-273
- applying**
 - activities, 43-45
 - animation, 110-115
 - applications, servers, 199-200
 - AsyncTask class, 206
 - Autocomplete, 293
 - Button control, 156
 - colors, 57-58
 - dialog boxes, 48-49
 - dimensions, 58-59
 - drawable resources, 59-60
 - EditText control, 156
 - emulators, 36
 - files, 64-66, 136-137
 - raw, 64-65
 - XML, 64
 - fragments, 49-50
 - GridView control, 124-128
 - handlers, 207
 - HTTP, 203-204
 - images, 59-60
 - intents, 46-48
 - layouts, 60-64
 - logging, 50-51

- resources, applications, 53-56
- Service objects, 218-219
- Spinner control, 159
- strings, 57
- threads, 207
- ViewSwitcher controls, 186-190
- XML, 146-147
- Appstore (Amazon), 244**
- arrays, 66**
 - strings, adding resources, 153
- assertTrue() method, 257**
- AsyncTask class, 206**
 - applying, 206
 - book downloads, extending for, 213
 - scores, downloading, 207-208
- attributes**
 - applications, 77, 200
 - backgrounds, 122
 - inheriting, 136
 - inputType, 156
 - maxLength, 156
 - maxLines, 156
 - minLines, 156
 - rank, 146
 - score, 146
 - titles, 129
 - username, 146
 - XML, TextView control, 136
- audio, 64**
- author's website, accessing, 301-303**
- Autocomplete, applying, 293**
- automation**
 - builds, 12
 - imports, 294. *See also* imports
 - testing, 253-260
- available space, 280**
- AVD (Android Virtual Device), 20**
 - managing, 20-21

B

- backgrounds**
 - attributes, 122
 - managing, 206
 - processing, 215
- batches of books, parsing, 213-216**
- best practices, testing applications, 249-252**
- Bitmap objects, 190**

blocks

- code, formatting, 294
- try/catch, 137

blogs, 31**BookListDownloaderTask class, starting, 215****books, parsing, 213-216****boolean values, 66****bugs, reporting, 287. See also debugging****builds**

- applications, 20-28, 39
 - activities, 43-45
 - context, 42-43
 - design, 39-41
 - dialog boxes, 48-49
 - fragments, 49-50
 - intents, 46-48
 - logging, 50-51
 - testing, 250
- automation, 12
- errors, troubleshooting, 297
- files, 15
- processes, 250
- release candidates, 263-265
- screens, tabs, 144-145
- targets, selecting, 13
- tracking, 74

Button control, 149

- applying, 156
- configuring, 157
- items, viewing, 182

buttons

- clicks, 158-159, 193
- Copy, 35
- Create AVD, 21
- Debug, 23
- Delete, 33
- Menu, 130
- Pull, 33
- Push, 33
- Refresh, 35
- Rotate, 34
- Screen Capture, 34
- Stop Process, 33
- Update Heap, 33
- Update Threads, 33

C**callbacks, activities, 44-46****calls, deprecated dialog, 165****cancel() method, 214****candidate builds, releases, 263-265****Canvas, 110****cases**

- edge, 195
- testing, 255

case statements, 169**CharacterPickerDialog, 166****checking**

- network status, 203
- for updates, 284

checklists, prerelease, 263**classes**

- Activity, 15, 40-41. *See also* Activity class
 - customizing dialogs, 171-177
 - DatePickerDialog class, 168-171
 - layout files, 40
 - lifecycles of dialogs, 167-168
 - methods, 48
- android.util.log, 50
- AsyncTask, 206
 - applying, 206
 - downloading scores, 207-208
 - extending book downloads, 213
- BookListDownloaderTask, starting, 215
- Dialog, 165-166
- factory, implementing, 187-189
- formatting, 293
- Fragment, 49
- getIntent(), 47
- Handler, 206
- Hashtable, 181, 192
- Log, 50
- PreferenceFragment, 151
- ProgressDialog, 166, 205
- ScoreDownloaderTask, 212
- Thread, 201, 206
- UploaderService, 219
- UploadTask, implementing, 220
- URL, 203
- UUID, 216

clicks, buttons, 158-159

driving games forward, 193

clients, 220. *See also* networks

testing, 251

closing

activities, 45

tabs, 291

cloud-monitoring solutions, 228**code**

Amazon Web Services SDK, 226

code signing tools, 12

formatting, 294

Java, writing, 293-297

refactoring, 294-296

reorganizing, 296

resources, storing, 54

standards, developing, 250

versions, settings, 74

collecting user input, 149, 165. *See also* user input**colors, 54**

applying, 57-58

resources, adding, 153

commands

emulators, 36

Organize Imports, 293

communication protocols, accessing, 199**compatibility, development**

environments, 280

components

Android SDKs, downloading, 282-284

games, 181. *See also* games

configuring

applications, 81

manifest files, 73-77

preferences, 93-95

AVDs, 20-21

Button control, 157

debugging, 22, 286

development

devices, 286-287

environments, 279-280

machine prerequisites, 280

EditText control, 156

manifest files, 69

network permissions, 203

resolution, 21

Spinner control, 160-161

tab defaults, 145

TabHost control, 144

targets, 27

XML, 146-147

conformance, testing, 252**connecting**

devices, 199

Mac operating systems, 285

networks, 201. *See also* networks

slow connections, 206

Windows operating systems, 285

consistency of screen design, 133**contacting authors, 303****contains() method, 163****containsKey() method, 193****context**

activities, launching, 43

applications, 42-43

menus, 128

controls

Button, 149

applying, 156

configuring, 157

DatePicker, 166

EditText, 149, 256

applying, 156

configuring, 156

listening for keystrokes, 175-177

forms, 155-161

FrameLayout, 102, 139

GridView, 117, 237

adding, 122-123

applying, 124-128

ImageButton, 157

ImageSwitcher, 181-182, 189-190

ImageView, 60, 102, 118

interfaces, formatting, 58

layouts, 66, 102-104

LinearLayout, 61, 102, 139, 150

Pick Date, 158

pleaseWaitDialog, 205

ProgressBar, 166, 201, 204-205

RelativeLayout, 103, 122, 135, 182

ScrollView, 133, 150

Spinner, 149, 153

applying, 159

configuring, 160-161

splash screens, 102

- TabHost, 142
 - adding, 139
 - configuring, 144
 - tabs, 145
- TableLayout, 102, 139
- TabSpec, 145
- TextSwitcher, 181, 182, 189
- TextView, 61, 63, 92, 102, 118, 136
- TimePicker, 166
- View, 181, 182
- ViewSwitcher
 - applying, 186-190
 - implementing factory classes, 187-189
- Copy button, 35**
- copyrights, DRM, 265**
- costs, 9-12**
- coverage, maximizing testing, 252-260**
- Create AVD button, 21**
- createFromResource() method, 154**
- creating. *See* configuring; formatting**
- currencies, 247**
- customizing**
 - applications, 237-240
 - layout controls, 66
 - log filters, 292

D

- Dalvik Debug Monitor Service. *See* DDMS**
- databases, 199. *See also* storing**
 - Amazon Web Services SDK, 226, 227
- Date of Birth setting, 149**
- DatePickerDialog, 166**
- dates, formatting, 246**
- DDMS (Dalvik Debug Monitor Service), 97**
 - applications, debugging, 31-35
 - perspective, 25
- Debug button, 23**
- Debug Configuration Manager, 23**
- debugging, 12, 252, 284-285**
 - applications, 20-28, 31-35, 95
 - configuring, 22, 286
 - diagnostics, 263
 - layouts with XML, 62-64
 - logging, 50
 - USB, 280
- declaring strings, literals, 192**

- defaults**
 - layouts, 61
 - locales, 242
 - resources, 236
 - tab settings, 145
- defect tracking systems, 251**
- defining**
 - activities, 77-79
 - color resources, 58
 - dialogs, 167
 - help screens, 87
 - identifier, 168
 - main menu screen features, 85-86
 - screens
 - game, 88
 - scores, 87
 - settings, 88
 - services, 218
 - SharedPreferences entries, 161
 - splash screen features, 85
- Delete button, 33**
- deleteFile() method, 136**
- deleting dialogs, 168**
- deployment**
 - applications, 12, 27
 - manual, 25
 - releases, 12
- deprecated dialog calls, 165**
- descriptions, applications, 76**
- design**
 - applications, 39-41
 - frameworks, 83
 - networks, 199-201
 - trivia games, 83-89
 - layouts
 - Layout Resource Editor, 61-62
 - XML, 62-64
 - main menu screens, 117-119
 - piracy, 264-265
 - prerelease checklists, 263
 - screens
 - games, 181-183
 - help, 133
 - scores, 138-139, 142-144
 - settings, 149-151
 - splash, 101
- determinate progress, 204**
- Developer.com, 304**
- Developer License Agreement, 272**

developers, 9

- accounts
 - benefits of, 273-274
 - registering, 271
- Amazon Appstore
 - Developer Portal, 304
 - registering as, 266
- anddev.org, 304
- Android Developer websites, 300, 304
- applications, registering, 261
- FierceDeveloper, 304
- Google Plus Android Developers, 304
- Java, 11
- OpenIntents, 304
- tools, 282
- websites, 282
- Wireless Developer Network, 304

Developer's Guide, 31**developing**

- applications, 11
 - networks, 201-202
 - release processes, 261-262
- code standards, 250
- costs, 9-12
- devices, configuring for, 286-287
- environments
 - configuring, 279-280
 - Eclipse. *See* Eclipse
 - IDEs, 280
- internationalizing, 240-241
- machine prerequisites, 280
- tools, 29
 - DDMS, 31-35
 - documentation, 29-31

Device Screen Capture tool, 34**devices, 10**

- applications, launching, 25-28
- AVDs, 20-21
- DDMS, 31-35
- debugging, 285
- development, configuring for, 286-287
- locales, 243
- networks, testing, 202
- screenshots, 34-35
- simulating, 252
- support, 233
- targets, testing on, 253

diagnostics, 35

- debugging, 263

dialects, 240**dialog boxes**

- Activity class, 165-168
 - customizing, 171-177
 - DatePickerDialog class, 168-171
 - lifecycles, 167-168
- applying, 48-49
- defining, 167
- deleting, 168
- dismissing, 168
- initializing, 167
- launching, 167
- New Android Project, 15
- passwords
 - launching, 177
 - layouts, 174
- progress
 - dismissing, 214
 - starting, 214
 - viewing, 204-205
- types of, 166-167
- user input, collecting, 165

Dialog classes, 165-166**digital rights management. *See* DRM****digital signatures, 267****Dimension resources, 184****dimensions, 54**

- applying, 58-59
- resources, 135, 184

directories, 136

- drawable resources, 61
- qualifiers, 235
- subdirectories, navigating, 18-19

dismiss() method, 205**dismissDialog() method, 48, 167****dismissing**

- dialogs, 168
- progress dialogs, 214

distribution, 9**docking modes, 233****docs subdirectory, 29-31****documentation, 29-31**

- emulators, 202
- Java, Amazon Web Services SDK, 226
- Javadoc, 297
- tabbing support, 139
- XML attributes, 136

doInBackground() method, 206, 210-211, 220

downloading

- Android SDK, 10
- batches of books, 213-216
- Eclipse, 281
- scores, 207-208, 224
- SDK, 282-284
- tools, 36

drawable resources, 184

- adding, 92
- applying, 59-60
- directories, 61

DRM (digital rights management), 265

E

ebook readers, 10**Eclipse, 12, 279**

- ADTs, installing, 282
- applications, testing, 253-260
- build files, 15
- DDMS, 31. *See also* DDMS
- debugging, 20
- installing, 281
- Layout Resource Editor, 61-62
- LogCat, 50
- managing, 289-293
- Manifest File Resource Editor, 69
- manifest files, editing, 17-18
- navigating, 12-19
- optimizing, 289
- preferences, 282
- tabs/windows outside of, 291
- updating, 284

edge cases, 195

- testing, 252

editing

- manifest files, 69
- resources
 - files, 18-19
 - projects, 17-19
- SharedPreferences, 162

editors

- Layout Resource Editor, 61-62
- Manifest File Resource Editor, 73
- resources, 12
- XML, 18

EditText control, 149, 256

- applying, 156
- configuring, 156
- keystrokes, listening for, 175-177

elements, <score>, 142**Email setting, 149****emulators, 12**

- applications
 - launching, 22-25
 - testing on, 252
- applying, 36
- DDMS, 31-35
- interaction, 31
- launching, 20
- managing, 25
- networks, testing, 202
- prototypes, launching, 95
- screenshots, 34-35
- SD cards, 20
- tabs, viewing, 142

enabling

- development settings, 286-287
- network support, 201
- Snapshot, 21

English, 242**entries, defining SharedPreferences, 161****environments**

- development
 - configuring, 279-280
 - IDEs, 280
- Eclipse. *See* Eclipse
- JRE, 281

errors

- build, troubleshooting, 297
- logging, 40
- text, viewing, 153

events, 44

- GridView control, 127-128
- lifecycles, animation, 114
- logging, 50

exceptions, NullPointerException, 142**execute() method, 212, 215****existing projects, adding, 15****exporting package files, 267-268****Extensible Markup Language. *See* XML****Extract Local Variable tool, 295****Extract Method tool, 295**

F

factory classes, implementing, 187-189

features

- applications, networks, 201
- DDMS, 31
- design, 39-41
- game screens, 182. *See also* games
- high-level game, 83-84
- main menu screens, defining, 85-86
- tracking, 74

feedback, sending, 303

fields, Minimum SDK, 15

fileList() method, 136

files

- applying, 64-66, 136-137
- file systems, navigating, 33
- graphics, 54
- JAR, 281
- jar, 15
- layouts, 54, 60-64
 - activity classes, 40
 - game.xml, 185
 - settings.xml, 154
 - XML, 49
- main menu screens, updating, 121-123
- managing, 31
- manifest, 17-18. *See also* manifest files
 - configuring, 69
 - launching activities, 43
 - navigating, 69-73
 - releases, 263-264
- packages, exporting, 267-268
- projects
 - creating, 13-15
 - navigating, 15-17
- raw, 54, 64-65
- raw resource
 - accessing, 137
 - adding, 136
- resources
 - editing, 18-19
 - XML, 18
- sections, viewing, 291
- source control, 290
- subdirectories, navigating, 18-19

XML

- adding, 58
- applying, 64
- parsing, 146-147

filling GridView controls, 124-127

filters, customizing logs, 292

findViewById() method, 64, 137, 145

finish() method, 45, 115

FierceDeveloper, 304

folders, source code, 15

formatting

- applications, creating new projects, 90-91
- classes, 293
- code, 294
- colors, 57-58
- files, 64-66
- graphics, 60
- interface controls, 58
- layouts, 60-64
- methods, 293
- projects, 13-15
- settings screens, 149-151
- strings, 57, 246
- tab defaults, 145
- XML, 146-147

forms, controls, 155-161

forums, 10

Fragment class, 49

fragments

- applying, 49-50
- dialogs, 49

frame-by-frame animation, 110

FrameLayout control, 102, 139

frameworks, applications

- design, 83
- trivia games, 83-89

French, 242

frequency of testing, 253

full applications, implementing internationalization, 245-246

functionality

- applications
 - accessing, 44
 - implementing, 41
- networks, 201
- testing, 251
- top-level application, 42

G

game.xml layout file, 185

games. See also applications

- logic, implementing, 190-196
- options menus, adding, 129-130
- running, 95-97
- screens
 - defining, 88
 - design, 181-183
 - implementing, 183-186
 - updating, 239
- settings, 149
- trivia, 83-89

garbage collection, 33

Gender setting, 149

generating reports, Amazon Appstore, 274

German, 242

GET method, 220-223

get() method, 193

getApplicationContext() method, 42

getAttributeValue() method, 147

getBookCoverImageDrawable() method, 190

getColor() method, 58

getDimension() method, 59

getDir() method, 136

getDrawable() method, 60

getFilesDir() method, 136

getFilePath() method, 136

getIntent() method, 163

getIntent() method, 47, 128

getLong() method, 163

getResources() method, 42, 56

getSharedPreferences() method, 42

getString() method, 57, 163

getText() method, 156

getView() method, 126

GIF (Graphics Interchange Format), 60, 110

Google

- App Engine, 200
- Mobile Services, 253
- Open Handset Alliance, 9-10
- Plus Android Developers, 304
- TV, 10

graphical layout editors, 12

Graphical Layout tab, 123

graphics, 53

- adding, 120
- animation, 110

- files, 54
- formatting, 60
- subdirectories, 19

Graphics Interchange Format. See GIF

GridView control, 117, 237

- adding, 122-123
- applying, 124-128

guidelines, testing applications, 250

H

handleNoBooks() method, 196

Handler class, 206

handlers, applying, 207

Hashtable class, 181

- books, storing, 192

Have You Read That? application, 84

- game screens, 181
- help screens, 133-137
- main menu screens, 118
- menus, 128
- network support, 202
- options menus, 130
- scores screens, 133, 138-144
- servers, 200
- settings screen passwords, 172
- splash screens, 101, 126
- user input, collecting, 149

headers

- main menu screens, 118
- scores screens, adding, 147

HelloKindleDebug configuration, 23, 25

help screens, 40, 133

- defining, 87
- design, 133
- layouts
 - implementing, 135-136
 - updating, 135-136

HelpActivity class, 40, 47

hierarchies, alternative, 233-236

Hierarchy Viewer, 12

high-level game features, 83-84

HTML (Hypertext Markup Language), 29

HTTP (Hypertext Transfer Protocol), 199-200, 203-204

HttpClient package, 220

HTTPS (Hypertext Transfer Protocol Secure), 199

Hypertext Markup Language. See HTML

Hypertext Transfer Protocol. See HTTP
Hypertext Transfer Protocol Secure.
See HTTPS

I

icons

- applications, 76
- launching, 92
- Settings, 202

identifiers

- defining, 168
- validating, 216

IDEs (integrated development environments), 10, 280

ImageButton control, 157

images

- animation settings, 113-114
- applying, 59-60
- resources, programming, 60

ImageSwitcher control, 181-182, 189-190

ImageView control, 60, 102, 118

implementing

- animation, splash screens, 101
- applications
 - activities, 92-93
 - functionality, 41
 - internationalization, 245-246
 - prototypes, 90-95
- factory classes, 187-189
- game screens, 183-186
- help screen layouts, 135-136
- logic, applications, 190-196
- main menu screens, 117
- password dialog layouts, 174
- scores screens, 141-144
- settings screens layouts, 151-155
- splash screens, 102-109
- UploadTask class, 220

imports, managing, 293-294

indeterminate progress, 204

InformIT website, 301

infrastructure, Amazon Web Services SDK, 226, 228

inheriting attributes, 136

initializing

- DatePickerDialogs, 170
- dialogs, 167
- switchers, 187

input

- Activity class dialogs, 165-168
 - customizing, 171-177
 - DatePickerDialog class, 168-171
 - lifecycles, 167-168
- emulators, 36
- form controls, 155-161
- passwords, 172
- settings screens, implementing, 151-155
- SharedPreferences, 161-163
- text, 156
- users, collecting, 149

InputStreamToString() method, 137

inputType attribute, 156

insertScoreRow() method, 147

installing

- ADTs, 282
- Android SDK, 281-282
- Eclipse, 281
- JDK, 281
- signed application packages, 269

instrumentation, manifest files, 18

Instrumentation tab, 72

Int object, 44

integers

- arrays, 66
- values, 66

integrated development environments.

See IDEs

integrating

- source-control services, 289
- testing, 251

<intent-filter> tag, 79

Intent information, 130

intents, applying, 46-48

interfaces, 12, 53

- controls, formatting, 58
- Eclipse, managing, 289-293
- layout files, 19
- Layout Resource Editor, 61-62
- orienting, 233
- settings.xml layout file, 154
- trivia games, 83-84

internationalization, 54

- applications, 240-241
- full applications, implementing, 245-246
- strategies, 243-246

Internet, 199. See also networks

resources, 304

service providers. *See* ISPs

slow connections, 206

ISO 639-1, 242

ISPs (Internet service providers), 227

<item> tag, 129

items, viewing, 182

iteration, loops, 146

J

JAR files, 15, 281

Java, 11

code, writing, 293-297

Development Kit. *See* JDK

documentation, Amazon Web Services

SDK, 226

resource layouts, 241

Runtime Environment. *See* JRE

Server Pages. *See* JSP

Javadoc documentation, 297

JDK (Java Development Kit), 279, 281

Joint Photographic Experts Group. See JPEG

JPEG (Joint Photographic Experts Group), 60

JRE (Java Runtime Environment), 281

JSP (JavaServer Pages), 200, 207

batch requests, 213-216

JUnit, 253-260

K

keyboard input, emulators, 36

keystrokes, listening for, 175-177

Kindle Fire. See devices

L

landscape mode, 237

languages

HTML, 29

Java, 11

localization, 240-243

specific resources, 242

support, 233

XML, 18

launching

activities, 43-44, 78-79

applications, 22-25

applying intents, 47

devices, 25-28

custom password dialogs, 177

DatePickerDialogs, 170

dialogs, 167

emulators, 20

icons, 92

prototypes, 95

Layout Resource Editor, 61-62

layouts

applying, 60-64

controls, 66, 102-104

design

help screens, 133

Layout Resource Editor, 61-62

XML, 62-64

files, 54

activity classes, 40

game.xml, 185

settings.xml, 154

XML, 49

game screens, 183. *See also* games

help screens

implementing, 135-136

updating, 135-136

main menu screens, 117-119, 121-123

password dialogs, implementing, 174

resources

adding, 92

Java, 241

programming, 63

scores screens, 138

implementing, 141-144

updating, 142-144

settings screens, 151-155

splash screens, updating, 106-109

LBS (Location-Based Services), 201

leaks, managing, 250

libraries

Amazon Web Services SDK, 226

LVL, 253

License Verification Library. See LVL

licenses, Developer License Agreement, 272

lifecycles

animation, 114

callback activities, 46

Fragment class, 49

limitations

- of emulators, 36
- of internationalization, 244, 245

LinearLayout control, 61, 102, 139, 150

Linux operating systems, 280

listening for events, 127

literals, declaring strings, 192

locales

- adding, 241
- applications, 241-243
- devices, 243
- settings, 241
- systems, 246

localization, 54, 240-241

- applications, 241-243
- tools, 246-247

Location-Based Services. See LBS

locations, source code, 13

Log class, 50

LogCat, 12, 35, 50

- navigating, 292

logging, 31

- applying, 50-51
- viewing, 35

logic, applications

- adding, 181
- implementing, 190-196

logs, customizing filters, 292

loops, while(), 146

LVL (License Verification Library), 253, 264

M

Mac operating systems, 280

- connecting, 285
- debugging, 286

machine prerequisites, 280

main menu screens

- defining, 85-86
- design, 117-119
- GridView control, applying, 124-128
- implementing, 117
- resources, adding, 120-121
- types, 128-130
- updating, 121-123, 237-238

main.xml tab, 61

maintenance, 35

makeView() method, 187

managing

activities

- dialogs, 48
- state, 44-45

Amazon Web Services SDK, 226, 228

Application Manager, 96

application resources, 53, 236

AVDs, 20-21

backgrounds, 206

Debug Configuration Manager, 23

DRM, 265

Eclipse, 289-293

emulators, 25

files, 31

imports, 293-294

memory, 31, 250

networks, 200

qualifiers, 234-236

resources, 19, 104, 233-236

tasks, 31-32

upgrading, 74

windows, 291

manifest files, 17-18

activities

- defining, 77-79
- launching, 43

applications

- attributes, 77
- configuring, 73-77
- descriptions, 76
- icons, 76
- managing permissions, 79-81
- naming, 76
- settings, 81

configuring, 69

navigating, 69-73

releases, 263-264

updating, 93

Manifest tab, 70

<manifest> tag, 74

manual deployment, 25

master layouts, 122. See also layouts

maximizing

- testing coverage, 252-260
- windows, 290

maxLength attribute, 156

maxLines attribute, 156

measurements, dimensions, 59

media files, resources, 65

memory, managing, 31, 250**Menu button, 130****<menu> tag, 129****MenuActivity class, 40, 44****menus, 40, 66**

- context, 128
- options, 128-130
- types of, 128-130

messages

- Amazon Web Services SDK, 226-228
- logging, 50
- privacy, 202

methods

- addTab(), 145
- assertTrue(), 257
- callbacks, 44-46
- cancel(), 214
- contains(), 163
- containsKey(), 193
- createFromResource(), 154
- deleteFile(), 136
- dismiss(), 205
- dismissDialog(), 48, 167
- doInBackground(), 206, 210-211, 220
- execute(), 212, 215
- fileList(), 136
- findViewById(), 64, 137, 145
- finish(), 45, 115
- formatting, 293
- GET, 220-223
- get(), 193
- getApplicationContext(), 42
- getAttributeValue(), 147
- getBookCoverImageDrawable(), 190
- getColor(), 58
- getDimension(), 59
- getDir(), 136
- getDrawable(), 60
- getFilesDir(), 136
- getFilePath(), 136
- getInt(), 163
- getIntent(), 47, 128
- getLong(), 163
- getResources(), 42, 56
- getSharedPreferences(), 42
- getString(), 57, 163
- getText(), 156
- getView(), 126
- handleNoBooks(), 196

- inputStreamToString(), 137
- insertScoreRow(), 147
- Log class, 50
- makeView(), 187
- next(), 146
- onActivityResult(), 44
- onBind(), 219
- onCancelled(), 209
- onCreate(), 44, 78, 125, 161
 - Hashtable class, 193
 - initializing switchers, 187
- onCreateDialog(), 48, 167, 168
- onCreateOptionsMenu(), 130
- onCreateView(), 49
- onDestroy(), 44, 49
- onDestroyed(), 163
- onFinished(), 259
- onItemClick(), 128
- onOptionsItemSelected(), 130
- onPause(), 44, 49, 163, 219
- onPickDateButtonClick(), 158
- onPostExecute(), 206, 209, 214
- onPreExecute(), 206, 208, 214
- onPrepareDialog(), 48, 167, 168
- onProgressUpdate(), 206, 211
- onResume(), 44
- onSetPasswordButtonClick(), 158
- onStart(), 49
- onStartCommand(), 219
- onStop(), 49
- openFileInput(), 136
- openRawResource(), 64
- POST, 220
- publishProgress(), 206
- put(), 193
- putExtra(), 47
- randomUUID(), 216
- removeDialog(), 48, 167, 168
- setContent(), 145
- setContentView(), 49, 125
- setCurrentTabByTag(), 145
- setCurrentText(), 189
- setFactory(), 187
- setImageDrawable(), 189
- setInput(), 203
- setProgress(), 204
- setSelection(), 160
- setText(), 156
- setUp(), 256

- setup(), 145
- showDialog(), 48, 167
- startActivity(), 44, 46
- toString(), 156
- updateDate(), 170
- minimizing windows, 290**
- Minimum SDK field, 15**
- minimum SDK versions, 75-76**
- minLines attribute, 156**
- mobile devices, 199. See also devicesmock**
- score data, 141. See also scores**
- modes**
 - docking, 233
 - landscape, 237
- money, 247**
- monitoring, 12, 228**
- moving tabs, 290**
- MySQL, 200**

N

- naming**
 - applications, 76
 - packages, 74
 - projects, 13
 - versions, 74
- navigating**
 - Amazon Web Services SDK, 226-228
 - Eclipse, 12-19
 - emulators, 36
 - file systems, 33
 - manifest files, 69-73
 - project files, 15-17
 - prototypes, 96-97
 - subdirectories, 18-19
- networks**
 - activities, 201
 - applications
 - design, 199-201
 - developing, 201-202
 - batches of books, downloading, 213-216
 - emulators, testing, 202
 - Kindle Fire, testing, 202
 - managing, 200
 - permissions, configuring, 203
 - player data
 - synchronizing, 216-217
 - uploading, 216
 - ProgressBar control, 204-205

- remote servers, uploading settings data
 - to, 217-223
- resources, accessing, 80
- scores
 - downloading, 208-213, 224
 - uploading, 223
- servers, 141
- services, accessing, 202-204
- status, checking, 203
- support, 199, 202
- tasks, running asynchronously, 206-207
- New Android Project dialog box, 15**
- new projects, creating, 13-15**
- next() method, 146**
- Nickname setting, 149**
- nine-patch stretchable images, 60**
- notifications, 44**
 - Amazon Web Services SDK, 227-228
- NullPointerException, 142**

O

- obfuscation, 12**
- objects**
 - Bitmap, 190
 - Int, 44
 - releases, 250
 - Service, applying, 218-219
- onActivityResult() method, 44**
- onBind() method, 219**
- onCancelled() method, 209**
- onCreate() method, 44, 78, 125, 161**
 - Hashtable class, 193
 - switchers, initializing, 187
- onCreateDialog() method, 48, 167, 168**
- onCreateOptionsMenu() method, 130**
- onCreateView() method, 49**
- onDestroy() method, 44, 49**
- onDestroyed() method, 163**
- onFinished() method, 259**
- onItemClick() method, 128**
- online resources, 304**
- onOptionsItemSelected() method, 130**
- onPause() method, 44, 49, 163, 219**
- onPickDateButtonClicked() method, 158**
- onPostExecute() method, 206, 209**
 - progress dialogs, dismissing, 214
- onPreExecute() method, 206, 208**
 - progress dialogs, starting, 214

onPrepareDialog() method, 48, 167, 168
onProgressUpdate() method, 206, 211
onResume() method, 44
onSetPasswordButtonClick() method, 158
onStart() method, 49
onStartCommand() method, 219
onStop() method, 49
Open Handset Alliance, 9-10
open platforms, 9
OpenFileInput() method, 136
OpenGL, 110
OpenIntents, 304
openRawResource() method, 64
operating systems
 debugging, configuring, 286
 support, 280
optimizing
 code, 12
 Eclipse, 289
 scores screens, 147
options
 adding, 129-130
 menus, 128
org.apache.http.client package, 220
Organize Imports command, 293
orienting interfaces, 233

P

packages
 applications, 12, 267-270
 files, exporting, 267-268
 HttpClient, 220
 naming, 15, 74
 org.apache.http.client, 220
panes, Logcat, 292
parameters
 JSP pages, 213
 View, 158
parsing
 book data, 192-196
 books, 213-216
 XML
 declaring string literals, 192
 files, 146-147
passing information with intents, 47
Password setting, 149

passwords
 colors, 153
 dialogs
 launching, 177
 layouts, implementing, 174
 settings screens, 172
performance, testing, 252
permissions
 android.permission.INTERNET, 190
 android.permission.WAKE_LOCK, 203
 applications, managing, 79-81
 manifest files, 18
 networks, configuring, 203
Permissions tab, 71
perspectives, repositioning tabs, 290
PHP, 200
Pick Date controls, 158
piracy, protecting applications from, 264-265
pixels, configuring, 21
planning
 applications, testing, 251-252
 network support, 202
 publishing, 261
platforms
 Open Handset Alliance, 9
 overview of, 10-11
 resources, 54
play screens, 40
PlayActivity class, 40
player data
 synchronizing, 216-217
 uploading, 216, 220-223
pleaseWaitDialog control, 205
plug-ins, ADTs, 280. *See also* ADTs
 installing, 282
PNG (Portable Network Graphics), 60
Portable Network Graphics. *See* PNG
ports, HTTP, 203. *See also* connecting
positioning tabs, 290
POST method, 220
precedence, events, 44
PreferenceFragment class, 151
preferences
 applications
 accessing, 42-43
 configuring, 93-95
 Eclipse, 282
 retrieving, 220
 SharedPreferences, 161-163

prerelease checklists, 263
 primitives, 66
 privacy messages, 202
 processes
 backgrounds, 215
 builds, 250
 releases, 261-262
 programming
 alternative resources, 236
 development machine prerequisites, 280
 image resources, 60
 resources
 layouts, 63
 storing, 54
ProgressBar control, 166, 201
 network activities, 204-205
ProgressDialog class, 166, 205
ProGuard, 12, 15, 264-265
 projects
 applications
 adding resources, 91-92
 creating new, 90-91
 creating, 13-15
 existing, adding, 15
 files, navigating, 15-17
 resources
 adding, 135
 editing, 17-19
 searching, 292
 testing, 254
property animation, 110
protocols, communication, 199
prototypes
 applications
 implementing, 90-95
 running, 95-97
 launching, 95
publisher's websites, accessing, 300
publishing
 Amazon AppStore, selling on, 271-274
 applications, 271
 planning, 261
 royalties, 273
publishProgress() method, 206
Pull button, 33
Pull Parser (XML), 146
Push button, 33
put() method, 193
putExtra() method, 47

Q

qualifiers, managing, 234-236
 queries, application servers, 200
QuickFix, 296-297

R

randomUUID() method, 216
rank attribute, 146
rankings, 138. See also scores
raw files, 54
 applying, 64-65
 XML, editing out, 63
raw resource files
 accessing, 137
 adding, 136
raw sockets, 199
refactoring code, 294-296
references, 31
 application resources, 55-56
Refresh button, 35
regions, resources, 242-243
registering
 accounts, 266
 activities, 77-78
 applications, 261
 developers
 accounts, 271
 Amazon Appstore, 266
 release processes, 261-262
RelativeLayout control, 103, 122, 135, 182
releases
 candidate builds, 263-265
 deployment, 12
 manifest files, 263-264
 objects, 250
 prerelease checklists, 263
 processes, 261-262
 services, 265
remote access. See also accessing
remote servers, 200
 scores, uploading, 223
 settings, uploading, 217-223
removeDialog() method, 48, 167, 168
removing dialogs, 168
Rename tool, 294
RenderScript, 110

reorganizing code, 296. See also managing reporting bugs, 287

reports, Amazon Appstore, 274

repositioning tabs, 290

requests

batches, JSP, 213-216

HTTP, 200

resources, 234

requirements

help screens, 133

main menu screens, 118

scores screens layouts, 138

resetting, 290. See also settings

resolution, configuring, 21

Resource Chooser, 123

resources, 31

adding, 91-92

applications

applying, 53-56

managing, 53, 236

references, 55-56

retrieving, 42

storing, 54

values, 57-59

colors, adding, 153

dimensions, 184

drawable

adding, 92

applying, 59-60

directories, 61

editors, 12

files

editing, 18-19

XML, 18

images, programming, 60

layouts, 60-64

Java, 241

programming, 63

main menu screens, adding, 120-121

managing, 19, 104, 233-236

networks, accessing, 80

online, 304

projects

adding, 135

editing, 17-19

raw files, adding, 136-137

regions, 242-243

releases, 250

requests, 234

scores screens, 141-142

settings screens, 151

splash screens, adding, 103-106

strings, adding, 153

types of, 66

URIs, 48

XML, retrieving, 146

responding to events, 127

results, launching activities, 44

retrieving

application resources, 42

book data, 192-196

preferences, 42, 220

shared preferences, 94

XML resources, 146

reviewing

manifest files, 263

source code, 90

RIM BlackBerry, 9

Rotate button, 34

routines, parsing, 146

royalties, publishing, 273

rules, alternative resources, 234

running. See also builds

applications, 20-28

automated tests, 258

configuring, 22

prototypes, 95-97

tasks asynchronously, 206-207

runtime

JRE, 281

resources, accessing, 53

S

saving

activity state, 45

preferences, 42

settings to SharedPreferences, 161

SAX (Simple API for XML), 146

score attribute, 146

<score> element, 142

score screens, 40

<score> tag, 146

ScoreDownloaderTask class, 212

scores

- downloading, 207-208, 224
- screens, 133
 - defining, 87
 - design, 138-139
 - implementing, 141-144
 - layouts, 138
 - optimizing, 147
 - updating, 142-144
- servers, uploading, 223
- trivia games, 83
- viewing, 208-213

ScoresActivity class, 40**Screen Capture button, 34****screens. *See also* interfaces**

- dialogs, 48-49. *See also* dialog boxes
- games
 - defining, 88
 - design, 181-183
 - implementing, 183-186
 - updating, 239
- help, 133
 - defining, 87
 - design, 133
- main menu
 - defining, 85-86
 - design, 117-119
 - implementing, 117
 - updating, 237-238
- scores, 133
 - defining, 87
 - design, 138-139
 - implementing, 141-144
 - layouts, 138
 - optimizing, 147
 - updating, 142-144
- settings
 - customizing dialogs, 172-175
 - defining, 88
 - design, 149-151
 - implementing layouts, 151-155
 - passwords, 172
- splash, 40
 - trivia games, 85-86
 - updating, 238-239
- tabs, builds, 144-145
- trivia games, 83-84

screenshots, 34-35**scrolling, 133. *See also* navigating****ScrollView control, 133, 150****SD cards, emulators, 20****SDK (Software Development Kit), 9, 279**

- Amazon Web Services SDK, 225
 - navigating, 226-228
 - overview of, 225-226
- documentation, 30
- downloading, 282-284
- installing, 281-282
- Minimum SDK field, 15
- preview versions, 10
- qualifiers, applying, 236
- targets, selecting, 13
- troubleshooting, 287
- upgrading, 284
- versions, 75-76

searching projects, 292**sections, viewing files, 291****security**

- permissions, 79-81
- piracy, 264-265

selecting

- builds, targets, 13
- options menus, 130
- skins, 21

selling on Amazon Appstore, 271-274.***See also* publishing****sequences, animation, 66****servers**

- applications, applying, 199-200
- networks, 141
- remote, uploading settings data to, 217-223
- scores, uploading, 223
- testing, 251

Service objects, applying, 218-219**services**

- Amazon Web Services SDK, 225
 - navigating, 226-228
 - overview of, 225-226
- DRM, 265
- networks, 201-204
- releases, 265
- source-control, integrating, 289
- UploaderService class, 219

SES (Amazon Simple Email Service), 227**set-top boxes, 10**

- `setContent()` method, 145
- `setContentView()` method, 49, 125
- `setCurrentTabByTag()` method, 145
- `setCurrentText()` method, 189
- `setFactory()` method, 187
- `setImageDrawable()` method, 189
- `setInput()` method, 203
- `setProgress()` method, 204
- `setSelection()` method, 160
- `setText()` method, 156
- settings
 - animation, images, 113-114
 - applications, 81
 - code, formatting, 294
 - games, adding state, 191
 - locales, 241
 - manifest files, 73-77
 - options menus, 128
 - perspectives, 290
 - remote servers, uploading, 217-223
 - screens
 - customizing dialogs, 172-175
 - defining, 88
 - design, 149-151
 - implementing layouts, 151-155
 - passwords, 172
 - tabs, defaults, 145
 - version code, 74
 - XML, 146-147
- Settings icon, 202
- `settings.xml` layout file, 154
- `setUp()` method, 145, 256
- `SharedPreferences`, 161-163
 - game state settings, adding, 191
- sharing preferences, retrieving, 94
- `showDialog()` method, 48, 167
- shutting down activities, 45
- side-by-side, viewing windows, 291
- signing applications, 267-268
- Simple API for XML. *See* SAX
- simulating devices, 252
- sizing controls, 58
- skins, selecting, 21
- slow connections, Internet, 206
- smartphones, 10
- Snapshot, enabling, 21
- SNS (Amazon Simple Notification Service), 227

- software system requirements, 280. *See also* applications
- source code, 299-300. *See also* code
 - folder, 15
 - locations, 13
 - reviewing, 90
- source-control services, integrating, 289
- Spinner control, 149, 153
 - applying, 159
 - configuring, 160-161
- splash screens, 40
 - animation, 101
 - applying, 110-115
 - design, 101
 - implementing, 102-109
 - controls, 102
 - Have You Read That? application, 126
 - layouts, updating, 106-109
 - resources, adding, 103-106
 - trivia games, 85-86
 - updating, 238-239
- `SplashActivity` class, 40
- SQS (Amazon Simple Queue Service), 228
- standards, 9
 - applications, testing, 250
 - code, developing, 250
- `startActivity()` method, 44, 46
- starting
 - `BookListDownloaderTask` class, 215
 - progress dialogs, 214
- state
 - activities, managing, 44-45
 - games, adding settings, 191
 - networks, 203
- statements
 - case, 169
 - switch, 169
- status, checking networks, 203
- Stop Process button, 33
- storing
 - Amazon Web Services SDK, 227
 - application resources, 54
 - book data, 192-196
- strategies, internationalization, 243-246
- String resources, 184
- strings, 53-54
 - applying, 57
 - arrays, 66

- formatting, 246
- languages, specifying, 242
- literals, declaring, 192
- resources, adding, 91, 153
- scores screen, 141
- styles, 66. See also formatting**
 - Javadoc documentation, 297
- subclasses, 60. See also classes**
- subdirectories**
 - docs, 29-31
 - navigating, 18-19
- support**
 - colors, formatting, 58
 - devices, 233
 - images, 59-60
 - languages, 233
 - networks, 199, 202
 - operating systems, 280
 - ProGuard, 12
 - tabs, 139
 - versions, 15
 - XML, 146-147
- switch statements, 169**
- switchers, initializing, 187**
- synchronizing player data, 216-217**
- systems**
 - default tracking, 251
 - locales, 246
 - requirements
 - development machine prerequisites, 280
 - environments, 280
 - resources, applying, 53-56

T

- TabHost control, 142**
 - adding, 139
 - configuring, 144
 - tabs, adding, 145
- TableLayout control, 102, 139**
- tablets, 10**
- tabs**
 - adding, 145
 - AndroidManifest.xml, 73
 - Application, 71
 - Application Nodes section, 78
 - closing, 291
 - default settings, 145

- Graphical Layout, 123
- Instrumentation, 72
- main.xml, 61
- Manifest, 70
- manifest editor tabs, 17
- Manifest File Resource Editor, 69
- outside of Eclipse, 291
- Permissions, 71
- perspectives, repositioning, 290
- scores, 208
- screens, builds, 144-145
- support, 139
- viewing, 142

TabSpec control, 145

tags

- <application>, 76
- <item>, 129
- <manifest>, 74
- <intent-filter>, 79
- <score>, 146
- <scores>, 146
- <uses-permission>, 80, 203

targets

- applications, deployment, 27
- builds, selecting, 13
- devices, testing on, 253

tasks

- Eclipse, managing, 292-293
- managing, 31, 32
- networks, running asynchronously, 206-207

TCP/IP (Transmission Control Protocol/Internet Protocol), 199

templates, 61. See also layouts

- GridView control, adding, 122-123
- main menu screens, 121

testing

- applications, 249
 - best practices, 249-252
 - maximizing coverage, 252-260
 - packages, 268-270
- edge-case, 252
- networks
 - emulators, 202
 - Kindle Fire, 202
- prereleases, 263
- projects, 15

text

- errors, viewing, 153
- input, 156
- TextSwitcher control, 181-182**
 - updating, 189
- TextView control, 61, 63, 92, 102, 118**
 - XML attributes, 136
- themes, 66**
- Thread class, 201, 206**
- threads, applying, 207**
- time, formatting, 246**
- TimePickerDialog, 166**
- titles, attributes, 129**
- T-Mobile G1, 10**
- tools, 9, 282**
 - ADTs, installing, 282
 - Android Developer websites, 300
 - code signing, 12
 - development, 29
 - DDMS, 31-35
 - documentation, 29-31
 - Device Screen Capture, 34
 - downloading, 36, 282-284
 - emulators. *See* emulators
 - Extract Local Variable, 295
 - Extract Method, 295
 - Hierarchy Viewer, 12
 - localization, 246-247
 - LogCat, 12, 50
 - Rename, 294
 - upgrading, 284
 - XML, 64
- top-level application functionality, 42**
- toString() method, 156**
- tracking**
 - builds, 74
 - defect tracking systems, 251
 - features, 74
- transitions, activities, 165**
- Transmission Control Protocol/Internet Protocol. *See* TCP/IP**
- trivia games, 83-89**
 - running, 95-97
- troubleshooting, 35, 282**
 - Android SDK, 287
 - build errors, 297
 - cloud-monitoring solutions, 228
 - defect tracking systems, 251
 - main screens, 237

- networks, 203
- QuickFix, 296-297
- slow connections, 206

try/catch blocks, 137**typed arrays, 66****types**

- of animation, 110
- of dialogs, 166-167
- of drawable resources, 60
- of main menu screens, 128-130
- of resources, 53, 66
- of scores, 138

U

uniform resource identifiers. *See* URIs**Update Heap button, 33****Update Threads button, 33****updateDate() method, 170****updating**

- game screens, 185-186, 239
- ImageSwitcher control, 189-190
- layouts
 - help screen, 135-136
 - splash screens, 106-109
- main menu screens, 121-123, 237-238
- manifest files, 93
- scores screens, 142-144
- splash screens, 238-239
- TextSwitcher control, 189

upgrading

- Android SDK, 284
- managing, 74

UploaderService class, 219**uploading**

- applications, 272-273
- player data, 216, 220-223
- scores, servers, 223
- settings, remote servers, 217-223

UploadTask class, implementing, 220**URIs (uniform resource identifiers), 48****URL class, 203****usability testing, 251****USB (universal serial bus), debugging, 280****user input**

- Activity class dialogs, 165-168
 - customizing, 171-177
- DatePickerDialog class, 168-171
- lifecycles, 167-168
- types of, 166-167

- collecting, 149
- dialog boxes, 165
- form controls, 155-161
- settings screens
 - design, 149-151
 - implementing, 151-155
- SharedPreferences, 161-163
- username attribute, 146
- <users-permission> tag, 80, 203
- utilities. *See* tools
- UUID class, 216

V

- validating identifiers, 216
- values
 - application resources, 57-59
 - boolean, 66
 - integers, 66
 - preferences, 94
- verifying
 - LVL, 253
 - manifest file releases, 263
 - signed applications, 269
- versions
 - applications, testing, 250
 - builds, 250
 - Eclipse, 281
 - manifest files, 74-75
 - SDK, 75-76
 - support, 15
- videos, 31
- View controls, 181, 182
- View parameter, 158
- viewing
 - error text, 153
 - file sections, 291
 - items, 182
 - logging, 35
 - progress, 204-205
 - scores, 208-213
 - tabs, 142
 - windows side-by-side, 291
- views, animation, 110, 112
- ViewSwitcher controls
 - applying, 186-190
 - factory classes, implementing, 187-189

W

- warnings, logging, 50
- websites
 - Android Developer, 300, 304
 - author's, accessing, 301-303
 - developers, 282
 - InformIT, 301
 - publisher's, accessing, 300
- while() loops, 146
- Wi-Fi, 202. *See also* connecting
 - permissions, 203
- windows
 - managing, 291
 - maximizing, 290
 - minimizing, 290
 - outside of Eclipse, 291
 - side-by-side, viewing, 291
- Windows operating systems, 280
 - connecting, 285
 - debugging, 286
- Wireless Developer Network, 304
- wizards, Android Project, 12-13
- workspaces, Eclipse, 289-293
- writing code, Java, 293-297

X-Z

- XML (Extensible Markup Language), 18
 - animation, 111. *See also* animation
 - attributes, TextView control, 136
 - files
 - adding, 58
 - applying, 64
 - parsing, 146-147
 - game screens, implementing, 184-185
 - layouts
 - design, 62-64
 - files, 49, 60-64
 - parsing, declaring string literals, 192
 - Pull Parser, 146
 - resources, retrieving, 146
 - support, 146-147
- XmlResourceParser, 146-147

Where Are the Companion Content Files?



Thank you for purchasing
**Learning Android™ Application Programming
for the Kindle Fire™: A Hands-On Guide to
Building Your First Android Application**

Access to the code used in the book is available by following the steps below:

1. On your PC or MAC, open a web browser and go to this URL:
www.informit.com/title/9780321833976
Navigate to the Downloads tab to access the sample code.
2. Download the Zip file (or files) from the web site to your hard drive.
3. Unzip the files.

Please note that many of our companion content files can be very large, especially image and video files.

If you are unable to locate the files for this title by following the steps at left, please visit www.informit.com/about/contact_us, select “Digital Products Help,” and enter in the Comments box the URL from step 1. Our customer service representatives will assist you.

Note: The code is also available on the authors’ website at <http://androidbook.blogspot.com>.

The Professional and Personal Technology Brands of Pearson



Cisco Press



informIT

PEARSON IT Certification



QUE

SAMS

vmware PRESS