

Ronan Schwarz
Phil Dutson
James Steele
Nelson To

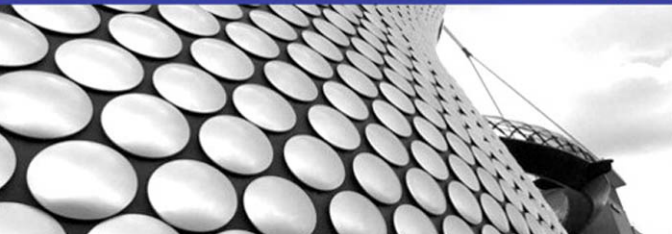


Second Edition

The Android™ Developer's Cookbook

Building Applications with the Android SDK

Developer's Library



FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Praise for *The Android™ Developer's Cookbook, Second Edition*

“The Android™ Developer's Cookbook, Second Edition, contains the recipes for developing and marketing a successful Android application. Each recipe in the book contains detailed explanations and examples of the right way to write your applications to become a featured app in the Google Play Store. From understanding the basic features of different versions of Android to designing and building a responsive UI, this cookbook gives you the recipes for success. You will learn to work with Android on every level—from hardware interfaces (like NFC and USB), to networking interfaces that will show you how to use mobile data efficiently, and even how to take advantage of Google's powerful billing interface. The authors do an incredible job of providing useful and real-life code examples for every concept in the book that can easily be built on and adapted to any situation and makes this book an essential resource for all Android developers.”

—David Brown, information data manager and application developer, San Juan School District

“Easy to read and easy to understand but not lacking features. This is one of the best books I have read on Android development. If you have the basics down, the recipes in the book will take you to mastery.”

—Casey Doolittle, lead Java developer, Icon Health and Fitness

“The Android™ Developer's Cookbook, Second Edition, provides a fantastic foundation for Android development. It teaches core skills such as layouts, Android life cycle, and responsiveness via numerous multi-threading techniques, which you need to be a skilled Android chef.”

—Kendell Fabricius, freelance Android developer

“This book has something for everyone. I've been programming Android since 1.0 and I learned some things that are completely new to me.”

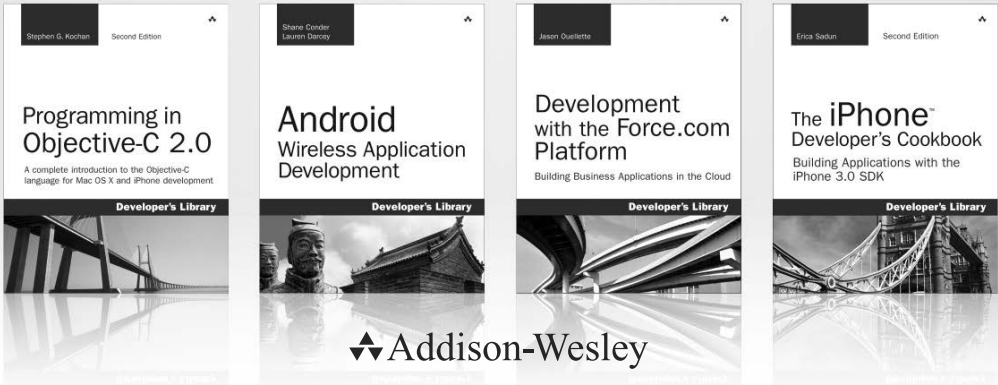
—Douglas Jones, senior software engineer, Fullpower Technologies

This page intentionally left blank

The Android™ Developer's Cookbook

Second Edition

Developer's Library Series



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

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

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

PEARSON

Addison-Wesley

Cisco Press

EXAMCRAM

IBM Press

QUE

PRENTICE HALL

SAMS

Safari

The Android™ Developer's Cookbook

Building Applications with
the Android SDK

Second Edition

Ronan Schwarz

Phil Dutson

James Steele

Nelson To

◆Addison-Wesley

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

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

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

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

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

For sales outside the United States, please contact:

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Ronan Schwarz,

The Android developer's cookbook : building applications with the Android SDK / Ronan Schwarz, Phil Dutson, James Steele, Nelson To.—Second edition.

pages cm

Includes index.

ISBN 978-0-321-89753-4 (pbk. : alk. paper)

1. Application software—Development. 2. Android (Electronic resource)
3. Operating systems (Computers) I. Schwarz, Ronan. II. Dutson, Phil,
1981– III. To, Nelson, 1976– IV. Title.

QA76.76.A65S743 2013
004.1675—dc23

2013014476

Copyright © 2013 Pearson Education, Inc.

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.

Google and the Google logo are registered trademarks of Google Inc., used with permission.

Android is a trademark of Google, Inc.

ISBN-13: 978-0-321-89753-4

ISBN-10: 0-321-89753-6

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

First printing, June 2013

Editor-in-Chief

Mark Taub

Executive Editor

Laura Lewin

Development Editor

Michael Thurston

Managing Editor

John Fuller

Project Editor

Elizabeth Ryan

Copy Editor

Barbara Wood

Indexer

Jack Lewis

Proofreader

Denise Wolber

Technical

Reviewers

Casey Doolittle

Douglas Jones

James Steele

Editorial Assistant

Olivia Basegio

Cover Designer

Chuti Prasertsith

Compositor

Acorn International



*To my beloved wife Susan and the OpenIntents Community:
Thank you for your support*

—Ronan

To Martin Simonnet and the Niantic Project for all the fun they have provided

—Phil

To Wei with love

—Jim

To my dear mom

—Nelson



This page intentionally left blank

Contents at a Glance

Preface **xxi**

About the Authors **xxv**

1	Overview of Android	1
2	Application Basics: Activities and Intents	21
3	Threads, Services, Receivers, and Alerts	51
4	Advanced Threading Techniques	89
5	User Interface Layout	109
6	User Interface Events	145
7	Advanced User Interface Techniques	177
8	Multimedia Techniques	199
9	Hardware Interface	221
10	Networking	251
11	Data Storage Methods	287
12	Location-Based Services	315
13	In-App Billing	343
14	Push Messages	349
15	Android Native Development	361
16	Debugging	371
A	Using the OpenIntents Sensor Simulator	395
B	Using the Compatibility Pack	401
C	Using a Continuous Integration System	409
D	Android OS Releases	411
	Index	417

This page intentionally left blank

Table of Contents

Preface **xxi**

About the Authors **xxv**

1	Overview of Android	1
	The Evolution of Android	1
	The Dichotomy of Android	2
	Devices Running Android	2
	HTC Models	3
	Motorola Models	5
	Samsung Models	5
	Tablets	5
	Other Devices	6
	Hardware Differences on Android Devices	6
	Screens	7
	User Input Methods	7
	Sensors	8
	Features of Android	10
	Multiprocess and App Widgets	10
	Touch, Gestures, and Multitouch	10
	Hard and Soft Keyboards	10
	Android Development	11
	Designing Applications Well	11
	Maintaining Forward Compatibility	11
	Ensuring Robustness	12
	Software Development Kit (SDK)	12
	Installing and Upgrading	12
	Software Features and API Level	14
	Emulator and Android Device Debug	14
	Using the Android Debug Bridge	15
	Signing and Publishing	16
	Google Play	16
	End User License Agreement	16
	Improving App Visibility	17
	Differentiating an App	18
	Charging for an App	18

Managing Reviews and Updates	19
Alternatives to Google Play	20
2 Application Basics: Activities and Intents	21
Android Application Overview	21
Recipe: Creating a Project and an Activity	22
Directory Structure of Project and Autogenerated Content	24
Android Package and Manifest File	26
Recipe: Renaming Parts of an Application	28
Recipe: Using a Library Project	29
Activity Lifecycle	31
Recipe: Using Activity Lifecycle Functions	31
Recipe: Forcing Single Task Mode	31
Recipe: Forcing Screen Orientation	34
Recipe: Saving and Restoring Activity Information	34
Recipe: Using Fragments	35
Multiple Activities	36
Recipe: Using Buttons and TextView	37
Recipe: Launching a Second Activity from an Event	38
Recipe: Launching an Activity for a Result Using Speech to Text	42
Recipe: Implementing a List of Choices	44
Recipe: Using Implicit Intents for Creating an Activity	45
Recipe: Passing Primitive Data Types between Activities	46
3 Threads, Services, Receivers, and Alerts	51
Threads	51
Recipe: Launching a Secondary Thread	52
Recipe: Creating a Runnable Activity	55
Recipe: Setting a Thread's Priority	56
Recipe: Canceling a Thread	57
Recipe: Sharing a Thread between Two Applications	57
Messages between Threads: Handlers	58
Recipe: Scheduling a Runnable Task from the Main Thread	58

Recipe: Using a Countdown Timer	60
Recipe: Handling a Time-Consuming Initialization	61
Alerts	63
Recipe: Using Toast to Show a Brief Message on the Screen	63
Recipe: Using an Alert Dialog Box	64
Recipe: Showing Notification in the Status Bar	65
Services	69
Recipe: Creating a Self-Contained Service	70
Recipe: Adding a WakeLock	74
Recipe: Using a Foreground Service	77
Recipe: Using an IntentService	80
Broadcast Receivers	82
Recipe: Starting a Service When the Camera Button Is Pressed	83
App Widgets	85
Recipe: Creating an App Widget	85
4 Advanced Threading Techniques	89
Loaders	89
Recipe: Using a CursorLoader	89
AsyncTasks	91
Recipe: Using an AsyncTask	92
Android Inter-Process Communication	94
Recipe: Implementing a Remote Procedure Call	94
Recipe: Using Messengers	99
Recipe: Using a ResultReceiver	105
5 User Interface Layout	109
Resource Directories and General Attributes	109
Recipe: Specifying Alternate Resources	111
Views and ViewGroups	112
Recipe: Building Layouts in the Eclipse Editor	113
Recipe: Controlling the Width and Height of UI Elements	115
Recipe: Setting Relative Layout and Layout ID	119
Recipe: Declaring a Layout Programmatically	120
Recipe: Updating a Layout from a Separate Thread	121

Text Manipulation	124
Recipe: Setting and Changing Text Attributes	124
Recipe: Providing Text Entry	127
Recipe: Creating a Form	129
Other Widgets: From Buttons to Seek Bars	130
Recipe: Using Image Buttons in a Table Layout	130
Recipe: Using Check Boxes and Toggle Buttons	134
Recipe: Using Radio Buttons	137
Recipe: Creating a Spinner	138
Recipe: Using a Progress Bar	140
Recipe: Using a Seek Bar	141
6 User Interface Events	145
Event Handlers and Event Listeners	145
Recipe: Intercepting a Physical Key Press	145
Recipe: Building Menus	148
Recipe: Defining Menus in XML	152
Recipe: Creating an Action Bar	154
Recipe: Using ActionBarSherlock	156
Recipe: Using the SEARCH Key	159
Recipe: Reacting to Touch Events	161
Recipe: Listening for Fling Gestures	163
Recipe: Using Multitouch	165
Advanced User Interface Libraries	168
Recipe: Using Gestures	168
Recipe: Drawing 3D Images	171
7 Advanced User Interface Techniques	177
Android Custom View	177
Recipe: Customizing a Button	177
Android Animation	183
Recipe: Creating an Animation	184
Recipe: Using Property Animations	187
Accessibility	189
Recipe: Using Accessibility Features	189
Fragments	191
Recipe: Displaying Multiple Fragments at Once	191
Recipe: Using Dialog Fragments	196

8 Multimedia Techniques 199

Images 199

Recipe: Loading and Displaying an Image for
Manipulation 202

Audio 206

Recipe: Choosing and Playing Back Audio Files 207

Recipe: Recording Audio Files 210

Recipe: Manipulating Raw Audio 211

Recipe: Using Sound Resources Efficiently 215

Recipe: Adding Media and Updating Paths 217

Video 217

Recipe: Using the VideoView 217

Recipe: Video Playback Using the MediaPlayer 219

9 Hardware Interface 221

Camera 221

Recipe: Customizing the Camera 222

Other Sensors 227

Recipe: Getting a Device's Rotational Attitude 227

Recipe: Using the Temperature and Light
Sensors 230

Telephony 231

Recipe: Using the Telephony Manager 232

Recipe: Listening for Phone States 234

Recipe: Dialing a Phone Number 235

Bluetooth 236

Recipe: Turning on Bluetooth 237

Recipe: Discovering Bluetooth Devices 237

Recipe: Pairing with Bonded Bluetooth Devices 238

Recipe: Opening a Bluetooth Socket 238

Recipe: Using Device Vibration 241

Recipe: Accessing the Wireless Network 241

Near Field Communication (NFC) 243

Recipe: Reading NFC Tags 243

Recipe: Writing NFC Tags 245

Universal Serial Bus (USB) 248

10 Networking 251

Reacting to the Network State 251

Recipe: Checking for Connectivity	251
Recipe: Receiving Connectivity Changes	253
Using SMS	255
Recipe: Autosending an SMS Based on a Received SMS	257
Using Web Content	263
Recipe: Customizing a Web Browser	263
Recipe: Using an HTTP GET	264
Recipe: Using HTTP POST	267
Recipe: Using WebViews	269
Recipe: Parsing JSON	271
Recipe: Parsing XML	273
Social Networking	275
Recipe: Reading the Owner Profile	275
Recipe: Integrating with Twitter	275
Recipe: Integrating with Facebook	284
11 Data Storage Methods	287
Shared Preferences	287
Recipe: Creating and Retrieving Shared Preferences	288
Recipe: Using the Preferences Framework	288
Recipe: Changing the UI Based on Stored Data	290
Recipe: Adding an End User License Agreement	294
SQLite Database	297
Recipe: Creating a Separate Database Package	297
Recipe: Using a Separate Database Package	300
Recipe: Creating a Personal Diary	303
Content Provider	306
Recipe: Creating a Custom Content Provider	308
File Saving and Loading	312
Recipe: Using AsyncTask for Asynchronous Processing	313
12 Location-Based Services	315
Location Basics	315
Recipe: Retrieving Last Location	317

Recipe: Updating Location Upon Change	318
Recipe: Listing All Enabled Providers	320
Recipe: Translating a Location to an Address (Reverse Geocoding)	322
Recipe: Translating an Address to a Location (Geocoding)	324
Using Google Maps	325
Recipe: Adding Google Maps to an Application	328
Recipe: Adding Markers to a Map	329
Recipe: Adding Views to a Map	333
Recipe: Setting Up a Proximity Alert	336
Using the Little Fluffy Location Library	337
Recipe: Adding a Notification with the Little Fluffy Location Library	338
13 In-App Billing	343
Google Play In-App Billing	343
Recipe: Installing Google's In-App Billing Service	344
Recipe: Adding In-App Billing to an Activity	345
Recipe: Listing Items for In-App Purchase	346
14 Push Messages	349
Google Cloud Messaging Setup	349
Recipe: Preparing for Google Cloud Messaging	349
Sending and Receiving Push Messages	351
Recipe: Preparing the Manifest	351
Receiving Messages	353
Recipe: Adding the BroadcastReceiver Class	353
Recipe: Adding the IntentService Class	354
Recipe: Registering a Device	356
Sending Messages	356
Recipe: Sending Text Messages	357
Recipe: Sending Messages with AsyncTask	358
15 Android Native Development	361
Android Native Components	361
Recipe: Using Java Native Interface	362
Recipe: Using the NativeActivity	364

16 Debugging 371

Android Test Projects	371
Recipe: Creating a Test Project	371
Recipe: Populating Unit Tests on Android	373
Recipe: Using Robotium	375
Eclipse Built-In Debug Tools	377
Recipe: Specifying a Run Configuration	377
Recipe: Using the DDMS	377
Recipe: Debugging through Breakpoints	380
Android SDK Debug Tools	380
Recipe: Starting and Stopping the Android Debug Bridge	380
Recipe: Using LogCat	381
Recipe: Using the Hierarchy Viewer	384
Recipe: Using TraceView	385
Recipe: Using lint	388
Android System Debug Tools	390
Recipe: Setting Up GDB Debugging	392
A Using the OpenIntents Sensor Simulator 395	
Setting Up the Sensor Simulator	395
Adding the Sensor Simulator to an Application	398
B Using the Compatibility Pack 401	
Android Support Packages	401
Adding the Support Library to a Project	408
C Using a Continuous Integration System 409	
D Android OS Releases 411	
Cupcake: Android OS 1.5, API Level 3, Released April 30, 2009	411
Donut: Android OS 1.6, API Level 4, Released September 15, 2009	411
Eclair: Android OS 2.0, API Level 5, Released October 26, 2009	412
Froyo: Android OS 2.2, API Level 8, Released May 20, 2010	412
Gingerbread: Android OS 2.3, API Level 9, Released December 6, 2010	412

Honeycomb: Android OS 3.0, API Level 11, Released
February 22, 2011 413

Ice Cream Sandwich: Android OS 4.0, API Level 14,
Released October 19, 2011 413

Jelly Bean: Android OS 4.1, API Level 16, Released
July 9, 2012 414

Index 417

This page intentionally left blank

Preface

Android is the fastest growing mobile operating system (OS). With more than 800,000 applications available in the Google Play store, the Android ecosystem is growing as well. There is enough diversity in device features and wireless carriers to appeal to just about anyone.

Netbooks have always been a natural platform to adopt Android, but the liveliness behind Android has fed the growth further into tablets, televisions, and even automobiles. Many of the world's largest corporations—from banks to fast food chains to airlines—have established a presence in Android and offer compatible services. Android developers have many opportunities, and relevant apps reach more people than ever before, increasing the satisfaction of creating a relevant app.

Why an Android Cookbook?

The Android OS is simple to learn, and Google provides many libraries to make it easy to implement rich and complex applications. The only aspect lacking, as mentioned by many in the Android developer community, is clear and well-explained documentation. The fact that Android is open source means anyone can dive in and reverse engineer some documentation. Many developer bulletin boards have excellent examples that were deduced using exactly this method. Still, a book that has a consistent treatment across all areas of the OS is useful.

In addition, a clear working example is worth a thousand words of documentation. Developers faced with a problem usually prefer to do a form of extreme programming; that is, they find examples of working code that does something close to the solution and modify or extend it to meet their needs. The examples also serve as a way to see the coding style and help to shape other parts of the developer's code.

This Android cookbook fills a need by providing a variety of self-contained recipes. As each recipe is introduced, the main concepts of the Android OS are also explained.

Who Should Read This Book?

Users who are writing their own Android applications will get the most out of this cookbook. Basic familiarity with Java and the Eclipse development environment is assumed but not required for the majority of the book. Java is a modular language, and

most (if not all) of the example recipes can be incorporated with minimal change into the reader's own Android project. The motivation and coverage of each topic in this book make it usable as an Android course supplement.

Using the Recipes

In general, the code recipes in this cookbook are self-contained and include all the information necessary to run a working application on an Android device. Chapters 1 and 2 give an introduction to the overall use of Android, but feel free to jump around and start using whatever is necessary.

This book is written first as a reference, providing knowledge mostly by example with the greatest benefits through implementation of the recipes of interest. The main technique introduced in each recipe is specified in the section heading. However, additional techniques are included in each recipe as needed to support the main recipe.

After reading this book, a developer should

- Be able to write an Android Application from scratch
- Be able to write code that works across multiple versions of Android
- Be able to use the various Application Programming Interfaces (APIs) provided in Android
- Have a large reference of code snippets to quickly assimilate into applications
- Appreciate the various ways to do the same task in Android and the benefits of each
- Understand the unique aspects of Android programming techniques

Book Structure

- Chapter 1, “Overview of Android,” provides an introduction to all aspects of Android outside of the code itself. It is the only chapter that doesn't include recipes, but it provides useful background material.
- Chapter 2, “Application Basics: Activities and Intents,” provides an overview of the four Android components and an explanation of how an Android project is organized. It also focuses on the activity as a main application building block.
- Chapter 3, “Threads, Services, Receivers, and Alerts,” introduces background tasks such as threads, services, and receivers, as well as notification methods for these background tasks using alerts.
- Chapter 4, “Advanced Threading Techniques,” covers using AsyncTasks and using loaders.
- Chapter 5, “User Interface Layout,” covers the user interface screen layout and views.

- Chapter 6, “User Interface Events,” covers user-initiated events such as touch events and gestures.
- Chapter 7, “Advanced User Interface Techniques,” covers creating a custom view, using animation, offering accessibility options, and working with larger screens.
- Chapter 8, “Multimedia Techniques,” covers multimedia manipulation and record and playback of audio and video.
- Chapter 9, “Hardware Interface,” introduces the hardware APIs available on Android devices and how to use them.
- Chapter 10, “Networking,” discusses interaction outside of the Android device with SMS, web browsing, and social networking.
- Chapter 11, “Data Storage Methods,” covers various data storage techniques available in Android, including SQLite.
- Chapter 12, “Location-Based Services,” focuses on accessing the location through various methods such as GPS and using services such as the Google Maps API.
- Chapter 13, “In-App Billing,” provides an instruction set on including in-app billing in your application using Google Play services.
- Chapter 14, “Push Messages,” covers how to use GCM for handling push messages with an application.
- Chapter 15, “Native Android Development,” discusses the components and structure used for native development.
- Chapter 16, “Debugging,” provides the testing and debugging framework useful throughout the development cycle.

Additional References

There are many online references for Android. A few essential ones are

- Android Source Code: <http://source.android.com/>
- Android Developer Pages: <http://developer.android.com/>
- Open Source Directory: <http://osdir.com/>
- Stack Overflow Discussion Threads: <http://stackoverflow.com/>
- Talk Android Developer Forums: www.talkandroid.com/android-forums/

This page intentionally left blank

About the Authors

Ronan “Zero” Schwarz is cofounder of OpenIntents, a Europe-based open source company specializing in Android development. Ronan has more than fifteen years of programming experience in a wide variety of fields such as augmented reality, web, robotics, and business systems, as well as different programming languages, including C, Java, and Assembler. He has been working on the Android Platform since 2007 and, among other things, has helped create SplashPlay and Droidspray, both top finalists of the Google Android Developer Challenge I and II.

Phil Dutson is the lead UX and mobile developer for ICON Health and Fitness. He has worked on projects and solutions for NordicTrack, ProForm, Freemotion, Sears, Costco, Sam’s Club, and others. Through the years he has been using, tweaking, and writing programs for mobile devices from his first Palm Pilot 5000 to his current collection of iOS and Android devices. Phil has also authored *jQuery*, *jQuery UI*, and *jQuery Mobile*; *Sams Teach Yourself jQuery Mobile in 24 Hours*; and *Creating QR and Tag Codes*.

James Steele was doing postdoctoral work in physics at MIT when he decided to join a start-up in Silicon Valley. Fifteen years later he continues to innovate, bringing research projects to production in both the consumer and mobile markets. He actively presents at and participates in various Silicon Valley new technology groups. Jim is VP of Engineering at Sensor Platforms.

Nelson To has more than ten applications of his own in the Android Market. He has also worked on enterprise Android applications for Think Computer, Inc. (PayPhone), AOL (AIM), Stanford University (Education App), and Logitech (Google TV). He also assists in organizing the Silicon Valley Android Meetup Community and teaches Android classes in both the Bay Area and China.

This page intentionally left blank

Networking

Network-based applications provide increased value for a user, in that content can be dynamic and interactive. Networking enables multiple features, from social networking to cloud computing.

This chapter focuses on the network state, short message service (SMS), Internet resource-based applications, and social networking applications. Knowing the network state is important to applications that fetch or update information that is available through a network connection. SMS is a communication service component that enables the exchange of short text messages between mobile phone devices. Internet resource-based applications rely on web content such as HTML (HyperText Markup Language), XML (eXtensible Markup Language), and JSON (JavaScript Object Notation). Social networking applications, such as Twitter, are important methods for people to connect with each other.

Reacting to the Network State

Knowing how and if a device is connected to a network is a very important facet of Android development. Applications that stream information from a network server may need to warn users about the large amount of data that may be charged to their accounts. Application latency issues may also be a concern. Making some simple queries enables users to find out if they are currently connected through a network device and how to react when the connection state changes.

Recipe: Checking for Connectivity

The `ConnectivityManager` is used for determining the connectivity of a device. This recipe can be used to determine what network interfaces are connected to a network. Listing 10.1 uses the `ConnectivityManager` to display if the device is connected via Wi-Fi or Bluetooth.

Listing 10.1 `src/com/cookbook/connectivitycheck/MainActivity.java`

```

package com.cookbook.connectivitycheck;

import android.app.Activity;
import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
    TextView tv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.tv_main);
        try {
            String service = Context.CONNECTIVITY_SERVICE;
            ConnectivityManager cm = (ConnectivityManager) getSystemService(service);
            NetworkInfo activeNetwork = cm.getActiveNetworkInfo();

            boolean isWiFi = activeNetwork.getType() == ConnectivityManager.TYPE_WIFI;
            boolean isBT = activeNetwork.getType() == ConnectivityManager.TYPE_BLUETOOTH;

            tv.setText("WiFi connected: "+isWiFi+"\nBluetooth connected: "+isBT);
        } catch (Exception nullPointerException) {
            tv.setText("No connected networks found");
        }
    }
}

```

Listing 10.1 uses the constants `TYPE_WIFI` and `TYPE_BLUETOOTH` to check for connectivity on these networks. In addition to `TYPE_WIFI` and `TYPE_BLUETOOTH`, the following constants can also be used to determine connectivity:

- `TYPE_DUMMY`—For dummy data connections
- `TYPE_ETHERNET`—For the default Ethernet connection
- `TYPE_MOBILE`—For the default mobile data connection
- `TYPE_MOBILE_DUN`—For DUN-specific mobile data connections
- `TYPE_MOBILE_HIPRI`—For high-priority mobile data connections
- `TYPE_MOBILE_MMS`—For an MMS-specific mobile data connection
- `TYPE_MOBILE_SUPL`—For an SUPL-specific mobile data connection
- `TYPE_WIMAX`—For the default WiMAX data connection

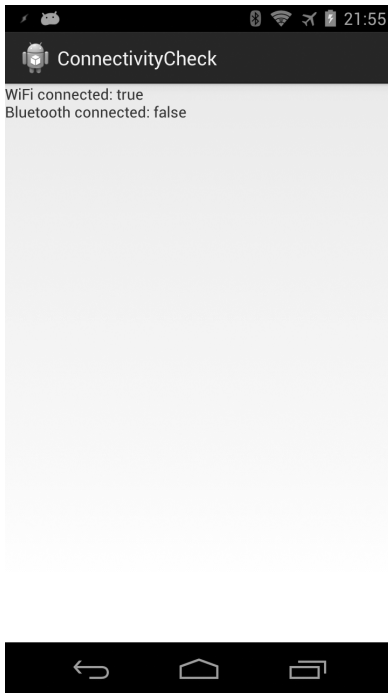


Figure 10.1 Checking for device connectivity

Figure 10.1 shows an application running with the code from Listing 10.1. Even though Bluetooth has been enabled, it reports false for being connected because it does not currently have an active connection.

Recipe: Receiving Connectivity Changes

A broadcast receiver can be used to check the status of network connectivity when it is necessary to react to changes in connectivity status.

A broadcast receiver can be declared in the application manifest, or it can be a subclass inside the main activity. While both are accessible, this recipe uses a subclass in conjunction with the `onCreate()` and `onDestroy()` methods to register and unregister the receiver.

As this recipe checks for connectivity, the following permissions need to be added to the application manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Listing 10.2 shows the code needed to check for connectivity changes. When a change is detected, the application will display a toast message informing the user of the change.

Listing 10.2 `src/com/cookbook/connectivitychange/MainActivity.java`

```
package com.cookbook.connectivitychange;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Bundle;
import android.widget.Toast;

public class MainActivity extends Activity {

    private ConnectivityReceiver receiver = new ConnectivityReceiver();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
        receiver = new ConnectivityReceiver();
        this.registerReceiver(receiver, filter);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (receiver != null) {
            this.unregisterReceiver(receiver);
        }
    }

    public class ConnectivityReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            ConnectivityManager conn =
                (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = conn.getActiveNetworkInfo();

            if (networkInfo != null && networkInfo.getType() == ConnectivityManager.
                TYPE_WIFI) {
                Toast.makeText(context, "WiFi is connected", Toast.LENGTH_SHORT).show();
            } else if (networkInfo != null) {
                Toast.makeText(context, "WiFi is disconnected", Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

```

    } else {
        Toast.makeText(context, "No active connection", Toast.LENGTH_SHORT).show();
    }
}
}
}

```

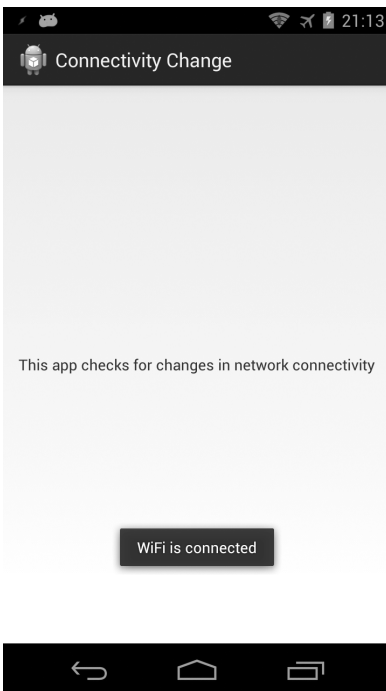


Figure 10.2 When Wi-Fi is enabled, a toast message appears informing the user of the connection

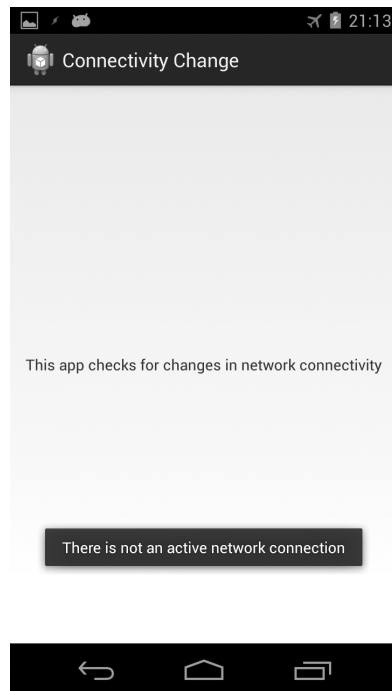


Figure 10.3 Wi-Fi and mobile data are disabled, so a toast informing the user of the lack of network connectivity is displayed

Figure 10.2 shows the message that appears when Wi-Fi is connected. Figure 10.3 shows the message that appears when both Wi-Fi and mobile data have been disconnected.

Using SMS

The Android Framework provides full access to SMS functionality using the `SmsManager` class. Early versions of Android placed `SmsManager` in the `android.telephony.gsm` package. Since Android 1.5, where `SmsManager` supports

both GSM and CDMA mobile telephony standards, the `SmsManager` class is now placed in the `android.telephony` package.

Sending an SMS through the `SmsManager` class is fairly straightforward:

1. Set the permission in the **AndroidManifest.xml** file to send SMS:

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

2. Use the `SmsManager.getDefault()` static method to get an SMS manager instance:

```
SmsManager mySMS = SmsManager.getDefault();
```

3. Define the destination phone number and the message that is to be sent. Use the `sendTextMessage()` method to send the SMS to another device:

```
String destination = "16501234567";
String msg = "Sending my first message";
mySMS.sendTextMessage(destination, null, msg, null, null);
```

This is sufficient to send an SMS message. However, the three additional parameters in the previous call set to `null` can be used as follows:

- The second parameter is the specific SMS service center to use. Set this to `null` to use the default service center from the carrier.
- The fourth parameter is a `PendingIntent` to track if the SMS message was sent.
- The fifth parameter is a `PendingIntent` to track if the SMS message was received.

To use the fourth and fifth parameters, a sent message and a delivered message intent need to be declared:

```
String SENT_SMS_FLAG = "SENT_SMS";
String DELIVER_SMS_FLAG = "DELIVER_SMS";

Intent sentIn = new Intent(SENT_SMS_FLAG);
PendingIntent sentPIn = PendingIntent.getBroadcast(this,0,sentIn,0);

Intent deliverIn = new Intent(SENT_SMS_FLAG);
PendingIntent deliverPIn
    = PendingIntent.getBroadcast(this,0,deliverIn,0);
```

Then, a `BroadcastReceiver` class needs to be registered for each `PendingIntent` to receive the result:

```
BroadcastReceiver sentReceiver = new BroadcastReceiver(){
    @Override public void onReceive(Context c, Intent in) {
        switch(getResultCode()){
            case Activity.RESULT_OK:
                //sent SMS message successfully;
```

```

        break;
    default:
        //sent SMS message failed
        break;
    }
}

};

BroadcastReceiver deliverReceiver = new BroadcastReceiver(){
    @Override public void onReceive(Context c, Intent in) {
        //SMS delivered actions
    }
};

registerReceiver(sentReceiver, new IntentFilter(SENT_SMS_FLAG));
registerReceiver(deliverReceiver, new IntentFilter(DELIVER_SMS_FLAG));

```

Most SMSs are restricted to 140 characters per text message. To make sure the message is within this limitation, use the `divideMessage()` method that divides the text into fragments in the maximum SMS message size. Then, the method `sendMultipartTextMessage()` should be used instead of the `sendTextMessage()` method. The only difference is the use of an `ArrayList` of messages and pending intents:

```

ArrayList<String> multiSMS = mySMS.divideMessage(msg);
ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
ArrayList<PendingIntent> deliverIns = new ArrayList<PendingIntent>();

for(int i=0; i< multiSMS.size(); i++){
    sentIns.add(sentIn);
    deliverIns.add(deliverIn);
}

mySMS.sendMultipartTextMessage(destination, null,
                                multiSMS, sentIns, deliverIns);

```

Recipe: Autosending an SMS Based on a Received SMS

Because most SMS messages are not read by the recipient until hours later, this recipe sends an autoresponse SMS when an SMS is received. This is done by creating an Android service in the background that can receive incoming SMSs. An alternative method is to register a broadcast receiver in the **AndroidManifest.xml** file.

The application must declare permission to send and receive SMSs in the **AndroidManifest.xml** file, as shown in Listing 10.3. It also declares a main activity `SMSResponder` that creates the autoresponse and a service `ResponderService` to send the response when an SMS is received.

Listing 10.3 AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.SMSResponder"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".SMSResponder"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:enabled="true" android:name=".ResponderService">
        </service>
    </application>

    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
</manifest>

```

The main layout file shown in Listing 10.4 contains a `LinearLayout` with three views: a `TextView` to display the message used for the autoreponse, `Button` used to commit changes on the reply message inside the application, and `EditText` where the user can enter a reply message.

Listing 10.4 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:id="@+id/display"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:textSize="18dp"
    />
    <Button android:id="@+id/submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Change my response"
    />
    <EditText android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />
</LinearLayout>

```

The main activity is shown in Listing 10.5. It starts the service that listens and auto-responds to SMS messages. It also allows the user to change the reply message and save it in `SharedPreferences` for future use.

Listing 10.5 `src/com/cookbook/SMSresponder/SMSResponder.java`

```
package com.cookbook.SMSresponder;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class SMSResponder extends Activity {
    TextView tv1;
    EditText ed1;
    Button bt1;
    SharedPreferences myprefs;
    Editor updater;
    String reply=null;

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

        myprefs = PreferenceManager.getDefaultSharedPreferences(this);
        tv1 = (TextView) this.findViewById(R.id.display);
        ed1 = (EditText) this.findViewById(R.id.editText);
        bt1 = (Button) this.findViewById(R.id.submit);

        reply = myprefs.getString("reply",
            "Thank you for your message. I am busy now."
            + "I will call you later");
        tv1.setText(reply);

        updater = myprefs.edit();
        ed1.setHint(reply);
        bt1.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                updater.putString("reply", ed1.getText().toString());
                updater.commit();
                SMSResponder.this.finish();
            }
        });
    }

    try {
```

```

        // Start service
        Intent svc = new Intent(this, ResponderService.class);
        startService(svc);
    }
    catch (Exception e) {
        Log.e("onCreate", "service creation problem", e);
    }
}
}

```

The majority of code is contained in the service, as shown in Listing 10.6. It retrieves `SharedPreferences` for this application first. Then, it registers a broadcast receiver for listening to incoming and outgoing SMS messages. The broadcast receiver for outgoing SMS messages is not used here but is shown for completeness.

The incoming SMS broadcast receiver uses a bundle to retrieve the protocol description unit (PDU), which contains the SMS text and any additional SMS meta-data, and parses it into an Object array. The method `createFromPdu()` converts the Object array into an `SmsMessage`. Then the method `getOriginatingAddress()` can be used to get the sender's phone number, and `getMessageBody()` can be used to get the text message.

In this recipe, after the sender address is retrieved, the `respond()` method is called. This method tries to get the data stored inside `SharedPreferences` for the auto-respond message. If no data is stored, it uses a default value. Then, it creates two `PendingIntents` for sent status and delivered status. The method `divideMessage()` is used to make sure the message is not oversized. After all the data is managed, it is sent using `sendMultiTextMessage()`.

Listing 10.6 `src/com/cookbook/SMSresponder/ResponderService.java`

```

package com.cookbook.SMSresponder;

import java.util.ArrayList;

import android.app.Activity;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;

public class ResponderService extends Service {

```

```

//the action fired by the Android system when an SMS was received
private static final String RECEIVED_ACTION =
    "android.provider.Telephony.SMS_RECEIVED";
private static final String SENT_ACTION="SENT_SMS";
private static final String DELIVERED_ACTION="DELIVERED_SMS";

String requester;
String reply="";
SharedPreferences myprefs;

@Override
public void onCreate() {
    super.onCreate();
    myprefs = PreferenceManager.getDefaultSharedPreferences(this);

    registerReceiver(sentReceiver, new IntentFilter(SENT_ACTION));
    registerReceiver(deliverReceiver,
        new IntentFilter(DELIVERED_ACTION));

    IntentFilter filter = new IntentFilter(RECEIVED_ACTION);
    registerReceiver(receiver, filter);

    IntentFilter attemptedfilter = new IntentFilter(SENT_ACTION);
    registerReceiver(sender,attemptedfilter);
}

private BroadcastReceiver sender = new BroadcastReceiver(){
    @Override
    public void onReceive(Context c, Intent i) {
        if(i.getAction().equals(SENT_ACTION)) {
            if(getResultCode() != Activity.RESULT_OK) {
                String recipient = i.getStringExtra("recipient");
                requestReceived(recipient);
            }
        }
    }
};

BroadcastReceiver sentReceiver = new BroadcastReceiver() {
    @Override public void onReceive(Context c, Intent in) {
        switch(getResultCode()) {
            case Activity.RESULT_OK:
                //sent SMS message successfully;
                smsSent();
                break;
            default:
                //sent SMS message failed
                smsFailed();
                break;
        }
    }
};

public void smsSent() {
    Toast.makeText(this, "SMS sent", Toast.LENGTH_SHORT);
}

public void smsFailed() {
    Toast.makeText(this, "SMS sent failed", Toast.LENGTH_SHORT);
}

```

```

    }
    public void smsDelivered() {
        Toast.makeText(this, "SMS delivered", Toast.LENGTH_SHORT);
    }

    BroadcastReceiver deliverReceiver = new BroadcastReceiver() {
        @Override public void onReceive(Context c, Intent in) {
            //SMS delivered actions
            smsDelivered();
        }
    };

    public void requestReceived(String f) {
        Log.v("ResponderService","In requestReceived");
        requester=f;
    }

    BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context c, Intent in) {
            Log.v("ResponderService","On Receive");
            reply="";
            if(in.getAction().equals(RECEIVED_ACTION)) {
                Log.v("ResponderService","On SMS RECEIVE");

                Bundle bundle = in.getExtras();
                if(bundle!=null) {
                    Object[] pdus = (Object[])bundle.get("pdus");
                    SmsMessage[] messages = new SmsMessage[pdus.length];
                    for(int i = 0; i<pdus.length; i++) {
                        Log.v("ResponderService","FOUND MESSAGE");
                        messages[i] =
                            SmsMessage.createFromPdu((byte[])pdus[i]);
                    }
                    for(SmsMessage message: messages) {
                        requestReceived(message.getOriginatingAddress());
                    }
                    respond();
                }
            }
        }
    };

    @Override
    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
    }

    public void respond() {
        Log.v("ResponderService","Responding to " + requester);
        reply = myprefs.getString("reply",
            "Thank you for your message. I am busy now."
            + "I will call you later.");
        SmsManager sms = SmsManager.getDefault();
        Intent sentIn = new Intent(SENT_ACTION);
        PendingIntent sentPin = PendingIntent.getBroadcast(this,
            0,sentIn,0);
    }

```

```

Intent deliverIn = new Intent(DEIVERED_ACTION);
PendingIntent deliverPin = PendingIntent.getBroadcast(this,
                                                    0,deliverIn,0);

ArrayList<String> Msgs = sms.divideMessage(reply);
ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
ArrayList<PendingIntent> deliverIns =
    new ArrayList<PendingIntent>();

for(int i=0; i< Msgs.size(); i++) {
    sentIns.add(sentPin);
    deliverIns.add(deliverPin);
}

sms.sendMultipartTextMessage(requester, null,
                             Msgs, sentIns, deliverIns);
}

@Override
public void onDestroy() {
    super.onDestroy();
    unregisterReceiver(receiver);
    unregisterReceiver(sender);
}

@Override
public IBinder onBind(Intent arg0) {
    return null;
}
}

```

Using Web Content

To launch an Internet browser to display web content, the implicit intent `ACTION_VIEW` can be used as discussed in Chapter 2, “Application Basics: Activities and Intents,” for example:

```

Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://www.google.com"));
startActivity(i);

```

It is also possible for developers to create their own web browser by using `WebView`, which is a `View` that displays web content. As with any view, it can occupy the full screen or only a portion of the layout in an activity. `WebView` uses `WebKit`, the open source browser engine used in Apple’s Safari, to render web pages.

Recipe: Customizing a Web Browser

There are two ways to obtain a `WebView` object. It can be instantiated from the constructor:

```

WebView webview = new WebView(this);

```


Alternatively, a `WebView` can be used in a layout and declared in the activity:

```
WebView webView = (WebView) findViewById(R.id.webview);
```

After the object is retrieved, a web page can be displayed using the `loadURL()` method:

```
webView.loadUrl("http://www.google.com/");
```

The `WebSettings` class can be used to define the features of the browser. For example, network images can be blocked in the browser to reduce the data loading using the `setBlockNetworkImage()` method. The font size of the displayed web content can be set using the `setDefaultFontSize()` method. Some other commonly used settings are shown in the following example:

```
WebSettings webSettings = webView.getSettings();
webSettings.setSaveFormData(false);
webSettings.setJavaScriptEnabled(true);
webSettings.setSavePassword(false);
webSettings.setSupportZoom(true);
```

Recipe: Using an HTTP GET

Besides launching a browser or using the `WebView` widget to include a WebKit-based browser control in an activity, developers might also want to create native Internet-based applications. This means the application relies on only the raw data from the Internet, such as images, media files, and XML data. Just the data of relevance can be loaded. This is important for creating social networking applications. Two packages are useful in Android to handle network communication: `java.net` and `android.net`.

In this recipe, an HTTP GET is used to retrieve XML or JSON data (see www.json.org/ for an overview). In particular, the Google search Representational State Transfer (REST) API is demonstrated, and the following query is used:

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=
```

To search for any topic, the topic just needs to be appended to the query. For example, to search for information on the National Basketball Association (NBA), the following query returns JSON data:

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=NBA
```

The activity needs Internet permission to run. So, the following should be added to the **AndroidManifest.xml** file:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

The main layout is shown in Listing 10.7. It has three views: `EditText` for user input of the search topic, `Button` to trigger the search, and `TextView` to display the search result.

Listing 10.7 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
    />
    <Button
        android:id="@+id/submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Search"
    />
    <TextView
        android:id="@+id/display"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello"
        android:textSize="18dp"
    />
</LinearLayout>
```

The main activity is shown in Listing 10.8. It initiates the three layout elements in `onCreate()`. Inside the `OnClickListener` class for the button, it calls `searchRequest()`. This composes the search item using the Google REST API URL and then initiates a URL class instance. The URL class instance is then used to get an `URLConnection` instance.

The `URLConnection` instance can retrieve the status of the connection. When `URLConnection` returns a result code of `HTTP_OK`, it means the whole HTTP transaction went through. Then, the JSON data returned from the HTTP transaction can be dumped into a string. This is done using an `InputStreamReader` passed to a `BufferedReader` to read the data and create a `String` instance. After the result from HTTP is obtained, it uses another function `processResponse()` to parse the JSON data.

Listing 10.8 src/com/cookbook/internet/search/GoogleSearch.java

```
package com.cookbook.internet.search;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
```

```

import java.net.URL;
import java.security.NoSuchAlgorithmException;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

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

public class GoogleSearch extends Activity {
    /** called when the activity is first created */
    TextView tv1;
    EditText ed1;
    Button bt1;
    static String url =
"http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv1 = (TextView) this.findViewById(R.id.display);
        ed1 = (EditText) this.findViewById(R.id.editText);
        bt1 = (Button) this.findViewById(R.id.submit);

        bt1.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                if(ed1.getText().toString()!=null) {
                    try{
                        processResponse(
                            searchRequest(ed1.getText().toString()));
                    } catch(Exception e) {
                        Log.v("Exception Google search",
                            "Exception:"+e.getMessage());
                    }
                }
                ed1.setText("");
            }
        });
    }

    public String searchRequest(String searchString)
        throws MalformedURLException, IOException {
        String newFeed=url+searchString;
        StringBuilder response = new StringBuilder();
        Log.v("gsearch","gsearch url:"+newFeed);
        URL url = new URL(newFeed);

        HttpURLConnection httpconn
            = (HttpURLConnection) url.openConnection();

```

```

        if(httpconn.getResponseCode()==URLConnection.HTTP_OK) {
            BufferedReader input = new BufferedReader(
                new InputStreamReader(httpconn.getInputStream()),
                8192);
            String strLine = null;
            while ((strLine = input.readLine()) != null) {
                response.append(strLine);
            }
            input.close();
        }
        return response.toString();
    }

    public void processResponse(String resp) throws IllegalStateException,
        IOException, JSONException, NoSuchAlgorithmException {
        StringBuilder sb = new StringBuilder();
        Log.v("gsearch","gsearch result:"+resp);
        JSONObject mResponseObject = new JSONObject(resp);
        JSONObject responseObject
            = mResponseObject.getJSONObject("responseData");
        JSONArray array = responseObject.getJSONArray("results");
        Log.v("gsearch","number of results:"+array.length());
        for(int i = 0; i<array.length(); i++) {
            Log.v("result",i+" "+array.get(i).toString());
            String title = array.getJSONObject(i).getString("title");
            String urllink = array.getJSONObject(i)
                .getString("visibleUrl");

            sb.append(title);
            sb.append("\n");
            sb.append(urllink);
            sb.append("\n");
        }
        tv1.setText(sb.toString());
    }
}

```

The detailed mechanism used requires an understanding of the incoming JSON data structure. In this case, the Google REST API provides all the result data under the results JSONArray. Figure 10.4 shows the search result for NBA.

Note that this recipe will run on Android projects only prior to API Level 11. This is due to running network requests on the main thread. The next recipe, “Using HTTP POST,” uses an AsyncTask to fix the NetworkOnMainThreadException that is thrown.

Recipe: Using HTTP POST

Sometimes, raw binary data needs to be retrieved from the Internet such as an image, video, or audio file. This can be achieved with the HTTP POST protocol by using `setRequestMethod()`, such as:

```
httpconn.setRequestMethod(POST);
```



Figure 10.4 The search result from the Google REST API query

Accessing data through the Internet can be time-consuming and unpredictable. Therefore, a separate thread should be spawned anytime network data is required.

In addition to the methods shown in Chapter 3, “Threads, Services, Receivers, and Alerts,” there is a built-in Android class called `AsyncTask` that allows background operations to be performed and publishes results on the UI thread without needing to manipulate threads or handlers. So, the `POST` method can be implemented asynchronously with the following code:

```
private class MyGoogleSearch extends AsyncTask<String, Integer, String> {

    protected String doInBackground(String... searchKey) {

        String key = searchKey[0];

        try {
            return searchRequest(key);
        } catch (Exception e) {
            Log.v("Exception Google search",
                "Exception:"+e.getMessage());
            return "";
        }
    }
}
```

```

protected void onPostExecute(String result) {
    try {
        processResponse(result);
    } catch (Exception e) {
        Log.v("Exception Google search",
            "Exception:"+e.getMessage());
    }
}
}

```

This excerpt can be added to the end of the **GoogleSearch.java** activity in Listing 10.8. It provides the same result with one additional change to the code inside the button `OnClickListener` to

```
new MyGoogleSearch().execute(ed1.getText().toString());
```

Recipe: Using WebViews

WebViews are useful for displaying content that may change on a semiregular basis, or for data that may need to be changed without having to force an update to the application. WebViews can also be used to allow web applications access to some client-side features of the Android system such as using the toast messaging system.

To add a WebView to an application, the following should be added to the layout XML:

```

<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

```

The following permission must also be added to the application manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
```

To create a simple page without any user interaction, add the following to the `onCreate()` method of the main activity:

```

WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com/");

```

In order to enable JavaScript on the page inside of the WebView, the `WebSettings` must be changed. This can be done using the following:

```

WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);

```

To trigger native methods from JavaScript, a class that can be used as an interface needs to be created. Listing 10.9 shows an activity with all of the pieces put together.

Listing 10.9 src/com/cookbook/viewtoaweb/MainActivity.java

```

package com.cookbook.viewtoaweb;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.webkit.JavascriptInterface;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.widget.Toast;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView myWebView = (WebView) findViewById(R.id.webview);
        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        myWebView.addJavascriptInterface(new WebAppInterface(this), "Android");
        myWebView.loadUrl("http://www.devcannon.com/androidcookbook/chapter10/webview/");
    }

    public class WebAppInterface {
        Context context;

        WebAppInterface(Context c) {
            context = c;
        }

        @JavascriptInterface
        public void triggerToast(String toast) {
            Toast.makeText(context, toast, Toast.LENGTH_SHORT).show();
        }
    }
}

```

The following HTML is used to trigger the code from Listing 10.9:

```

<input type="text" name="toastText" id="toastText" />
<button id="btn" onClick="androidToast()">Toast it</button>

```

The following JavaScript is used to trigger the code:

```

function androidToast() {
    var input = document.getElementById('toastText');
    Android.triggerToast(input.value);
}

```

Figure 10.5 displays the WebView with a toast that was launched from the page being viewed.

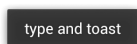
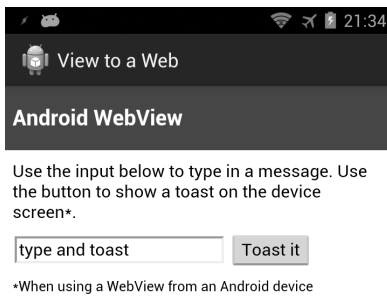


Figure 10.5 Triggering a toast message from a page inside a WebView

Recipe: Parsing JSON

JSON is a very popular format for data transfer, especially when used with web services. Android has included a set of classes in the `org.json` package that can be imported into code to allow manipulation of JSON data.

To get started parsing, first a JSON object needs to be created; this can be done like so:

```
private JSONObject jsonObject;
```

Some data in JSON format is also needed. The following creates a string containing some JSON data:

```
private String jsonString =
    "{\"item\":{\"name\":\"myName\",\"numbers\":[{\"id\":\"1\"},{\"id\":\"2\"}]}}\";
```

Because a string is not a JSON object, one will need to be created that contains the value of the string. This can be done like so:

```
jsonObject = new JSONObject(jsonString);
```

Now that there is an object to manipulate, data can be gotten from it. If the `getString()` method were used to pull data from an “object” that is inside the

`jsonObject`, a `JSONException` would be thrown. This is because it is not a string. To pull a specific value, another object must be set up that contains the desired string, like so:

```
JSONObject itemObject = jsonObject.getJSONObject("item");
```

The value of "name" can be gotten by using the following:

```
String jsonName = itemObject.getString("name");
```

A loop may be used to get the information stored in the "numbers" section of `jsonObject`. This can be done by creating a `JSONArray` object and looping through it, as follows:

```
JSONArray numbersArray = itemObject.getJSONArray("numbers");
```

```
for(int i = 0; i < numbersArray.length(); i++){
    numbersArray.getJSONObject(i).getString("id");
}
```

Listing 10.10 shows how parsing may be put together inside an activity and displayed in a `TextView`. Note that when pulling JSON data from a remote location, such as through a web service, a separate class or `AsyncTask` must be used so that the main UI thread is not blocked.

Listing 10.10 `src/com/cookbook/parsejson/MainActivity.java`

```
package com.cookbook.parsejson;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

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

public class MainActivity extends Activity {

    TextView tv;
    private JSONObject jsonObject;
    private String jsonString =
    "{\"item\":{\"name\":\"myName\",\"numbers\":{\"id\":\"1\"},{\"id\":\"2\"}}}\"";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.tv_main);

        try {
            jsonObject = new JSONObject(jsonString);
            JSONObject itemObject = jsonObject.getJSONObject("item");
            String jsonName = "name: " + itemObject.getString("name");
```

```
JSONArray numbersArray = itemObject.getJSONArray("numbers");
String jsonIds = "";

for(int i = 0;i < numbersArray.length();i++){
    jsonIds += "id: " +
        numbersArray.getJSONObject(i).getString("id").toString() + "\n";
}

tv.setText(jsonName+"\n"+jsonIds);

} catch (JSONException e) {
    e.printStackTrace();
}
}
```

Recipe: Parsing XML

The official Android documentation recommends the use of `XmlPullParser` for parsing XML data. You may use any method you prefer to get XML data; however, for this recipe, a simple one-node XML string will be used. Listing 10.11 shows an activity that will display the process of reading the XML document, including the node and text value, into a `TextView`.

The XML data is processed one line at a time, with the `next()` method moving to the next line. In order to parse for specific nodes inside the XML data, an `if else` statement must be added for them in the `while` loop.

Listing 10.11 `src/com/cookbook/parsexml/MainActivity.java`

```
package com.cookbook.parsexml;

import java.io.IOException;
import java.io.StringReader;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

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

public class MainActivity extends Activity {

    TextView tv;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv = (TextView) findViewById(R.id.tv_main);
    }
}
```

```

String xmlOut = "";
XmlPullParserFactory factory = null;
try {
    factory = XmlPullParserFactory.newInstance();
} catch (XmlPullParserException e) {
    e.printStackTrace();
}
factory.setNamespaceAware(true);
XmlPullParser xpp = null;
try {
    xpp = factory.newPullParser();
} catch (XmlPullParserException e) {
    e.printStackTrace();
}

try {
    xpp.setInput(new StringReader("<node>This is some text</node>"));
} catch (XmlPullParserException e) {
    e.printStackTrace();
}

int eventType = 0;
try {
    eventType = xpp.getEventType();
} catch (XmlPullParserException e) {
    e.printStackTrace();
}

while (eventType != XmlPullParser.END_DOCUMENT) {
    if(eventType == XmlPullParser.START_DOCUMENT) {
        xmlOut += "Start of XML Document";
    } else if (eventType == XmlPullParser.START_TAG) {
        xmlOut += "\nStart of tag: "+xpp.getName();
    } else if (eventType == XmlPullParser.END_TAG) {
        xmlOut += "\nEnd of tag: "+xpp.getName();
    } else if (eventType == XmlPullParser.TEXT) {
        xmlOut += "\nText: "+xpp.getText();
    }
    try {
        eventType = xpp.next();
    } catch (XmlPullParserException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
xmlOut += "\nEnd of XML Document";

tv.setText(xmlOut);
}
}

```

Social Networking

Twitter is a social networking and microblogging service that enables its users to send and read messages known as tweets. Twitter is described as the “SMS of the Internet,” and indeed, each tweet cannot exceed 140 characters (although links are converted to shorter links and not counted against the 140-character limit). Twitter users can follow other people’s tweets or be followed by others.

Recipe: Reading the Owner Profile

Starting with API Level 14 (Ice Cream Sandwich), developers are able to access the owner profile. This is a special contact that stores RawContact data. To read the owner profile of a device, the following permission must be added to the **AndroidManifest.xml** file:

```
<uses-permission android:name="android.permission.READ_PROFILE" />
```

The following enables access to profile data:

```
// sets the columns to retrieve for the owner profile - RawContact data
String[] mProjection = new String[]
{
    Profile._ID,
    Profile.DISPLAY_NAME_PRIMARY,
    Profile.LOOKUP_KEY,
    Profile.PHOTO_THUMBNAI_URI
};

// retrieves the profile from the Contacts Provider
Cursor mProfileCursor =
    getContentResolver().query(Profile.CONTENT_URI,mProjection,null,null,null);
// Set the cursor to the first entry (instead of -1)
boolean b = mProfileCursor.moveToFirst();
for(int i = 0, length = mProjection.length;i < length;i++) {
    System.out.println("*** " +
        mProfileCursor.getString(mProfileCursor.getColumnIndex(mProjection[i]));
}
```

Note that where `System.out.println()` is used is the place where logic can be inserted to process the profile information. It is also worth mentioning that the output will be shown in LogCat, even though it is not a method from `Log.*`.

Recipe: Integrating with Twitter

Some third-party libraries exist to assist in integrating Twitter into Android applications (from <http://dev.twitter.com/pages/libraries#java>):

- Twitter4J by Yusuke Yamamoto—An open source, Mavenized, and Google App Engine-safe Java library for the Twitter API, released under the BSD license
- Scribe by Pablo Fernandez—OAuth module for Java, Mavenized, and works with Facebook, LinkedIn, Twitter, Evernote, Vimeo, and more

For this recipe, the Twitter4J library by Yusuke Yamamoto is used, which has documentation at <http://twitter4j.org/en/javadoc/overview-summary.html>. The recipe enables users to log in to Twitter by using OAuth and make a tweet.

Twitter has made changes to its authentication system that now require applications to register in order to access the public feed. To get started, an application has to be registered at <https://dev.twitter.com/apps/new>. During the registration process, OAuth public and private keys will be generated. They will be used in this recipe, so take note of them.

As this application will be accessing the Internet, it will need the `INTERNET` permission. There will also be a check to make sure that the device is connected to a network, so the `ACCESS_NETWORK_STATE` permission is also required. This is done by editing the **AndroidManifest.xml** file, as shown in Listing 10.12.

Listing 10.12 AndroidManifest.xml

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

    <uses-sdk
        android:minSdkVersion="9"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.cookbook.tcookbook.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:scheme="oauth" android:host="tcookbook"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```

        </intent-filter>
    </activity>
</application>
</manifest>

```

For the layout of the application, everything will be put into the **activity_main.xml** file. This file will contain a button that is visible on page load and then several buttons, TextViews, and an EditText widget. Note that some of these will be hidden with `android:visibility="gone"`. Listing 10.13 shows the contents of the **activity_main.xml** file.

Listing 10.13 res/layout/activity_main.xml

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    tools:context=".MainActivity" >

    <Button android:id="@+id/btnLoginTwitter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Login with OAuth"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:layout_marginTop="30dip"/>

    <TextView android:id="@+id/lblUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dip"
        android:layout_marginTop="30dip"/>

    <TextView android:id="@+id/lblUpdate"
        android:text="Enter Your Tweet:"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dip"
        android:layout_marginRight="10dip"
        android:visibility="gone"/>

    <EditText android:id="@+id/txtUpdateStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:visibility="gone"/>

    <Button android:id="@+id/btnUpdateStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Tweet it!"
        android:layout_marginLeft="10dip"

```

```

        android:layout_marginRight="10dip"
        android:visibility="gone"/>

        <Button android:id="@+id/btnLogoutTwitter"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Logout/invalidate OAuth"
            android:layout_marginLeft="10dip"
            android:layout_marginRight="10dip"
            android:layout_marginTop="50dip"
            android:visibility="gone"/>
    </LinearLayout>

```

One activity is used in the application, and two classes are used: one to help with connection detection and one to display an alert message when the wrong application OAuth keys are used.

In the main activity, several constants are set up for use. These include the OAuth Consumer key and Consumer secret. A connectivity check is run to make sure that the user can reach Twitter. Several `OnClickListener` classes are also registered to trigger logic such as login, logout, and update when clicked.

As Twitter handles authentication for the user, the information passed back is saved in application preferences and is checked again when the user attempts to log in to the application. An `AsyncTask` is also used to move any tweets made to a background thread.

Listing 10.14 shows the contents of the activity in full.

Listing 10.14 `src/com/cookbook/tcookbook/MainActivity.java`

```

package com.cookbook.tcookbook;

import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import twitter4j.User;
import twitter4j.auth.AccessToken;
import twitter4j.auth.RequestToken;
import twitter4j.conf.Configuration;
import twitter4j.conf.ConfigurationBuilder;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.content.pm.ActivityInfo;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.os.StrictMode;
import android.text.Html;
import android.util.Log;

```

```

import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity {

    // Replace the following value with the Consumer key
    static String TWITTER_CONSUMER_KEY = "01189998819991197253";
    // Replace the following value with the Consumer secret
    static String TWITTER_CONSUMER_SECRET =
        "616C6C20796F75722062617365206172652062656C6F6E6720746F207573";

    static String PREFERENCE_NAME = "twitter_oauth";
    static final String PREF_KEY_OAUTH_TOKEN = "oauth_token";
    static final String PREF_KEY_OAUTH_SECRET = "oauth_token_secret";
    static final String PREF_KEY_TWITTER_LOGIN = "isTwitterLoggedIn";

    static final String TWITTER_CALLBACK_URL = "oauth://tcookbook";

    static final String URL_TWITTER_AUTH = "auth_url";
    static final String URL_TWITTER_OAUTH_VERIFIER = "oauth_verifier";
    static final String URL_TWITTER_OAUTH_TOKEN = "oauth_token";

    Button btnLoginTwitter;
    Button btnUpdateStatus;
    Button btnLogoutTwitter;
    EditText txtUpdate;
    TextView lblUpdate;
    TextView lblUserName;

    ProgressDialog pDialog;

    private static Twitter twitter;
    private static RequestToken requestToken;

    private static SharedPreferences mSharedPreferences;

    private ConnectionDetector cd;

    AlertDialogManager adm = new AlertDialogManager();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // used for Android 2.3+
        if (Build.VERSION.SDK_INT > Build.VERSION_CODES_GINGERBREAD) {
            StrictMode.ThreadPolicy policy =
                new StrictMode.ThreadPolicy.Builder().permitAll().build();
            StrictMode.setThreadPolicy(policy);
        }

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);

        cd = new ConnectionDetector(getApplicationContext());

```



```

    if (!cd.isConnectingToInternet()) {
        adm.showAlertDialog(MainActivity.this, "Internet Connection Error",
            "Please connect to working Internet connection", false);
        return;
    }

    if (TWITTER_CONSUMER_KEY.trim().length() == 0 ||
        TWITTER_CONSUMER_SECRET.trim().length() == 0) {
        adm.showAlertDialog(MainActivity.this,
            "Twitter OAuth tokens",
            "Please set your Twitter OAuth tokens first!", false);
        return;
    }

    btnLoginTwitter = (Button) findViewById(R.id.btnLoginTwitter);
    btnUpdateStatus = (Button) findViewById(R.id.btnUpdateStatus);
    btnLogoutTwitter = (Button) findViewById(R.id.btnLogoutTwitter);
    txtUpdate = (EditText) findViewById(R.id.txtUpdateStatus);
    lblUpdate = (TextView) findViewById(R.id.lblUpdate);
    lblUserName = (TextView) findViewById(R.id.lblUserName);

    mSharedPreferences = getApplicationContext().getSharedPreferences("MyPref", 0);

    btnLoginTwitter.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View arg0) {
            // Call login Twitter function
            loginToTwitter();
        }
    });

    btnUpdateStatus.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String status = txtUpdate.getText().toString();

            if (status.trim().length() > 0) {
                new updateTwitterStatus().execute(status);
            } else {
                Toast.makeText(getApplicationContext(),
                    "Please enter status message", Toast.LENGTH_SHORT).show();
            }
        }
    });

    btnLogoutTwitter.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View arg0) {
            // Call logout Twitter function
            logoutFromTwitter();
        }
    });

    if (!isTwitterLoggedInAlready()) {
        Uri uri = getIntent().getData();
    }

```

```

        if (uri != null && uri.toString().startsWith(TWITTER_CALLBACK_URL)) {
            String verifier = uri.getQueryParameter(URL_TWITTER_OAUTH_VERIFIER);

            try {
                AccessToken accessToken = twitter.getOAuthAccessToken(requestToken,
➡verifier);

                Editor e = mSharedPreferences.edit();

                e.putString(PREF_KEY_OAUTH_TOKEN, accessToken.getToken());
                e.putString(PREF_KEY_OAUTH_SECRET, accessToken.getTokenSecret());
                e.putBoolean(PREF_KEY_TWITTER_LOGIN, true);
                e.commit();

//                Log.e("Twitter OAuth Token", "> " + accessToken.getToken());

                btnLoginTwitter.setVisibility(View.GONE);

                lblUpdate.setVisibility(View.VISIBLE);
                txtUpdate.setVisibility(View.VISIBLE);
                btnUpdateStatus.setVisibility(View.VISIBLE);
                btnLogoutTwitter.setVisibility(View.VISIBLE);

                long userID = accessToken.getUserId();
                User user = twitter.showUser(userID);
                String username = user.getName();

                lblUserName.setText(Html.fromHtml("<b>Welcome " + username + "</b>"));
            } catch (Exception e) {
                Log.e("****Twitter Login Error: ",e.getMessage());
            }
        }
    }

    private void loginToTwitter() {
        if (!isTwitterLoggedInAlready()) {
            ConfigurationBuilder builder = new ConfigurationBuilder();
            builder.setOAuthConsumerKey(TWITTER_CONSUMER_KEY);
            builder.setOAuthConsumerSecret(TWITTER_CONSUMER_SECRET);
            Configuration configuration = builder.build();

            TwitterFactory factory = new TwitterFactory(configuration);
            twitter = factory.getInstance();

            if(!(Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)) {
                try {
                    requestToken = twitter.getOAuthRequestToken(TWITTER_CALLBACK_URL);
                    this.startActivity(new Intent(Intent.ACTION_VIEW,
                        Uri.parse(requestToken.getAuthenticationURL())));
                } catch (TwitterException e) {
                    e.printStackTrace();
                }
            } else {
                new Thread(new Runnable() {

```

```

        public void run() {
            try {
                requestToken = twitter.getOAuthRequestToken(TWITTER_CALLBACK_URL);
                MainActivity.this.startActivity(new Intent(Intent.ACTION_VIEW,
                    Uri.parse(requestToken.getAuthenticationURL())));
            } catch (TwitterException e) {
                e.printStackTrace();
            }
        }
    }).start();
}
} else {
    Toast.makeText(getApplicationContext(),"Already logged into Twitter",
        Toast.LENGTH_LONG).show();
}
}

class updateTwitterStatus extends AsyncTask<String, String, String> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        pDialog = new ProgressDialog(MainActivity.this);
        pDialog.setMessage("Updating to Twitter...");
        pDialog.setIndeterminate(false);
        pDialog.setCancelable(false);
        pDialog.show();
    }

    protected String doInBackground(String... args) {
//        Log.d("*** Text Value of Tweet: ",args[0]);
        String status = args[0];
        try {
            ConfigurationBuilder builder = new ConfigurationBuilder();
            builder.setOAuthConsumerKey(TWITTER_CONSUMER_KEY);
            builder.setOAuthConsumerSecret(TWITTER_CONSUMER_SECRET);

            String access_token =
                mSharedPreferences.getString(PREF_KEY_OAUTH_TOKEN, "");
            String access_token_secret =
                mSharedPreferences.getString(PREF_KEY_OAUTH_SECRET, "");

            AccessToken accessToken =
                new AccessToken(access_token, access_token_secret);
            Twitter twitter =
                new TwitterFactory(builder.build()).getInstance(accessToken);

            twitter4j.Status response = twitter.updateStatus(status);

//            Log.d("*** Update Status: ",response.getText());
        } catch (TwitterException e) {
            Log.d("*** Twitter Update Error: ", e.getMessage());
        }
        return null;
    }
}

```

```

protected void onPostExecute(String file_url) {
    pDialog.dismiss();
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(getApplicationContext(),
                "Status tweeted successfully", Toast.LENGTH_SHORT).show();
            txtUpdate.setText("");
        }
    });
}

private void logoutFromTwitter() {
    Editor e = mSharedPreferences.edit();
    e.remove(PREF_KEY_OAUTH_TOKEN);
    e.remove(PREF_KEY_OAUTH_SECRET);
    e.remove(PREF_KEY_TWITTER_LOGIN);
    e.commit();

    btnLogoutTwitter.setVisibility(View.GONE);
    btnUpdateStatus.setVisibility(View.GONE);
    txtUpdate.setVisibility(View.GONE);
    lblUpdate.setVisibility(View.GONE);
    lblUserName.setText("");
    lblUserName.setVisibility(View.GONE);

    btnLoginTwitter.setVisibility(View.VISIBLE);
}

private boolean isTwitterLoggedInAlready() {
    return mSharedPreferences.getBoolean(PREF_KEY_TWITTER_LOGIN, false);
}

protected void onResume() {
    super.onResume();
}
}

```

More information on using Twitter4j can be found in the following resources:

- www.androidhive.info/2012/09/android-twitter-oauth-connect-tutorial/ by Ravi Tamada
- <http://blog.doityourselfandroid.com/2011/08/08/improved-twitter-oauth-android/> by Do-it-yourself Android
- <http://davidcrowley.me/?p=410> by David Crowley
- https://tutsplus.com/tutorials/?q=true&filter_topic=90 by Sue Smith
- <http://blog.blundell-apps.com/sending-a-tweet/> by Blundell

Recipe: Integrating with Facebook

Facebook has changed rapidly in the last couple of years, and it remains one of the top social networking sites. One thing the Facebook team has done recently is to clean up their documentation to help developers. The official documentation can be found at <https://developers.facebook.com/docs/getting-started/facebook-sdk-for-android/3.0/>.

To get started with Facebook development, first download the Facebook SDK and the Facebook android package (APK) from <https://developers.facebook.com/resources/facebook-android-sdk-3.0.zip>. The APK is provided as a means of authentication without having to use a WebView. If the Facebook application is already installed on the phone, the APK file need not be installed.

Next, add the Facebook SDK as a library project to the Eclipse installation. This is done by choosing **File** → **Import** and then **General** → **Existing Projects into Workspace**. Note that Facebook warns against using the “Copy projects into workspace” options, as this may build incorrect filesystem paths and cause the SDK to function incorrectly.

After the Facebook SDK has been imported, the sample projects are available for experimentation. Note that most of the projects require the generation of a key hash that will be used to sign applications and that developers can add to their Facebook developer profile for quick SDK project access.

The key is generated by using the `keytool` utility that comes with Java. Open a terminal or command prompt and type the following to generate the key:

OS X:

```
keytool -exportcert -alias androiddebugkey -keystore ~/.android/debug.keystore |
[cc]openssl sha1 -binary | openssl base64
```

Windows:

```
keytool -exportcert -alias androiddebugkey -keystore %HOMEPATH%\android\debug.
keystore [ccc] openssl sha1 -binary | openssl base64
```

The command should be typed in a single line, although terminals or command prompt windows may show it breaking into multiple lines. When the command is executed, a password prompt should appear. The password to enter is **android**. After the key has been generated successfully, it will be displayed. Note that if a “‘keytool’ is not recognized as an internal or external command . . .” error is generated, move to the **bin** directory of the JRE installation directory and try again. If there is a similar error for “openssl,” download OpenSSL from <http://code.google.com/p/openssl-for-windows/>. If there are still errors, make sure that the **bin** directories have been added to the system path or that the exact directories are being used instead of **%HOMEPATH%**.

If more than one computer will be used for development, a hash must be generated for each one and added to the developer profile at <https://developers.facebook.com/>.

Once that is done, dig into the sample applications and log in with them. The showcase example project, called **HelloFacebookSample**, demonstrates how to access a profile, update a status, and even upload photos.

The last step in creating an application that integrates with Facebook is to create a Facebook app that will then be tied to the Android application by using a generated key hash. This will take care of integration and allow users to authenticate themselves while using the application.

The developer site gives a terrific breakdown of all the pieces needed to get started. Be sure to read the official Scrumptious tutorial, which can be found at <http://developers.facebook.com/docs/tutorials/androidsdk/3.0/scrumptious/>.

This page intentionally left blank

Index

A

AAC ELD (enhanced low-delay AAC), 200

AAC LC audio format, 200

Accelerometers

- accessibility of, 221
- determining device rotational attitude, 227–230
- screen orientation and, 34
- three-axis accelerometers, 9–10, 227–230

Accessibility

- checklist for, 189–190
- TalkBack and, 189–190
- using features of, 189–191

Accessory mode, USB devices and, 248–249

Action bars

- creating, 154–156
- example on device running Gingerbread, 158–159
- example on phone running Jelly Bean, 157
- example on tablet running Ice Cream Sandwich, 156

ActionBarSherlock

- bridging API levels prior to ver. 11, 154, 156–159
- using themes of, 158

Active-matrix organic LED (AMOLED) displays, 4, 7

Activities

- creating runnable activities, 55–56
- creating with Eclipse IDE, 22–24
- fragments of, 35–36

- multiple activities. *see* Multiple activities
- restoring activity information, 34–35
- saving relevant information, 34–35
- using loaders, 89–91

Activity lifecycle functions

- example of service lifecycle flowchart, 71
- flowchart, 32
- forcing screen orientation, 34
- forcing single task mode, 31–34
- restoring activity information, 34–35
- saving activity information, 34–35
- using fragments, 35–36
- using functions, 31
- using NativeActivity, 366–369

ADB. *see* Android Debug Bridge (ADB)

ADK. *see* Android Accessory Development Kit (ADK)

AdMob, 18, 19

ADT. *see* Android Development Tools (ADT)

ADT Bundle, 12–13, 371, 377

AIDL. *see* Android Interface Definition Language (AIDL)

AK8976A package (AKM), 9

Alert dialog boxes, for user options, 64–65

Alerts

- big-picture style notification, 67–68
- dialog boxes for user options, 64–65
- example of message alert, 51
- inbox-style notification, 69
- proximity alerts and Google Maps, 336

Alerts (continued)

- showing status bar pending notifications, 65–69
- using Toast to show brief screen message, 63–64

Amazon, 6**Amazon Appstore, 6, 20****Amazon MP3, 6****Amazon Video, 6****AMOLED displays, 4, 7****AMR-NB audio format, 200****Android, Inc., 1****Android Accessory Development Kit (ADK), 249****Android Asset Packaging Tool (aapt), 26****Android Beam, 243, 267, 414, 438****Android Debug Bridge (ADB)**

- accessing devices with, 15–16
- starting and stopping, 380–381
- using over wireless connection, 249

Android Development Tools (ADT)

- creating test suites, 2, 371
- downloading ADT Bundle, 12–13
- using lint tool with, 388–390

Android Interface Definition Language (AIDL)

- bridging between applications, 94
- data types supported by, 94
- example of output, 97
- implementing remote procedure call, 94–95
- RPC between processes with different user IDs, 99

Android Native Development Kit (NDK)

- activity lifecycle, 366–369
- app glue interfaces, 366–369
- building native library, 363
- downloading, 361
- example of output, 364
- initial steps, 361–362
- type mapping between Java and Native, 362
- using Java Native Interface, 362–364

- using NativeActivity, 364–369
- version 4, GDB debugging files, 363

Android operating system (OS), overview

- application design, 11
- aspects of SDK, 12–16
- devices, 7–8
- dichotomies of, 2
- evolution of, 1–2
- features of, 10–11
- Google Play, 16–20
- hardware differences, 6–10
- maintaining forward compatibility, 11–12
- robustness, 12
- support packages, 401–408
- types of devices, 2–6

Android OS Emulator Controls

- within DDMS, 380
- listing of, 15

Android OS releases, listing of

- Cupcake (Android OS 1.5, API Level 3, released 4/30/09), 411
- Donut (Android OS 1.6, API Level 4, released 9/15/09), 411
- Eclair (Android OS 2.0, API Level 5, released 10/26/09), 412
- Froyo (Android OS 2.2, API Level 8, released 5/20/10), 412
- Gingerbread (Android OS 2.3, API Level 9, released 12/6/10), 412–413
- Honeycomb (Android OS 3.0, API Level 11, released 2/22/11), 413
- Ice Cream Sandwich (Android OS 4.0, API Level 14, released 10/19/11), 413–414
- Jelly Bean (Android OS 4.1, API Level 16, released 7/9/12), 414–415

Android package, manifest file and, 26–28**Android Support Library, 156–157, 401–408****Android Virtual Devices (AVD)**

- emulator functions, 15
- managing, 325, 395

ANDROID-MK.HTML file, 363

Android.support.v4.accessibilityservice package, 401
Android.support.v4.app package, 402–403
Android.support.v4.content package, 404
Android.support.v4.content.pm package, 404
Android.support.v4.database package, 404
Android.support.v4.net package, 405
Android.support.v4.os package, 405
Android.support.v4.util package, 405
Android.support.v4.view package, 405–407
Android.support.v4.view.accessibility package, 407
Android.support.v4.widget package, 408
Animation
 advanced user interface techniques, 183–189
 creating mail animation, 184–186
 resource directories, 109
 using property animations, 187–189
ANR-WB audio format, 200
Apache Ant, 30, 409–410
Apache Continuum, 410
Apache License, 294–297, 409–410
Apache Maven, 156, 409–410
API key, 349, 358
App glue interfaces, 366–369
App Widgets. *see also* Standard graphical widgets
 and broadcast receivers, 85–87
 creating text display on home screen, 85–87
 Google’s design guidelines for, 11
 minimum update time, 85
 multiprocessing and, 10
 Views and ViewGroups and, 112–113
AppBrain, 20
Apple, Inc., 1
Application basics
 activity lifecycle functions, 31–36
 alerts, 63–69
 Android packages and manifest file, 26–28

 App Widgets, 85–87
 autogenerated content, 25–26
 broadcast receivers, 82–87
 components of application, 21, 22
 creating projects and activities, 22–24
 current context in anonymous inner class, 39
 directory structure, 24–26
 implementing list of choices, 44–45
 implicit intents for creating activities, 45–46
 launching activity for result using speech-to-text functionality, 42–44
 launching additional activity from event, 38–41, 42
 multiple activities, 36–49
 overview of, 21–22
 passing primitive data types between activities, 47–49
 renaming parts of application, 28–29
 services, 69–82
 threads, 51–58
 using buttons and TextView, 37–38
 using library projects, 29–31

Application design, 11

Application settings. *see* Settings

Archos, 5–7

Asahi Kasei Microsystems (AKM), 9

Asus, 6

AsyncTask

 advanced threading techniques, 91–93
 background operations and, 268–269
 pulling JSON data from remote locations, 272
 sending push messages with, 358–360
 using for asynchronous processing, 313–314

Attributes

 colors, 110–111
 dimensions, 110

Attributes (*continued*)

- EditText and text manipulation, 124, 127–128
- fonts, 110, 124–127
- string, 110
- TextView and text manipulation, 125

Audio

- adding media and updating paths, 217
- choosing and playing back audio files, 207–209
- frameworks for, 206
- manipulating raw audio, 211–215
- multimedia techniques, 206–217
- recording audio files, 210
- registering files to system, 217
- supported media types (Android 4.1), 200–201
- using HTTP POST to retrieve web data, 267–269
- using sound resources efficiently, 215–217

Auto-capitalization, text entry and, 129**Autogenerated content, project structure and, 25–26****Automobiles, Android systems and, 6****Autoreponse SMS, 257–263****AVD. *see* Android Virtual Devices (AVD)****AVD Manager, 13–15, 325, 395**

B

BACK key, KeyEvent and, 145–148**Backward compatibility, 12, 147****The Baidu App store, 20****Bamboo (CI system), 410****Battery power**

- broadcast receivers and, 82
- customer reviews and, 17
- Little Fluffy Location Library and, 337–341
- of Motorola phones, 5
- multiprocessing and, 10

- updating of widgets and, 85–87

- WakeLocks and, 74

Berne Convention, 16**Big-picture style notification alert, 67–68****Billing integration. *see* In-app billing (Google Play)****BitMapFactory, 199, 202–205****Bluetooth (BT)**

- accessing wireless networks, 241–242
- activating, 237
- checking for device connectivity to, 251–253
- discovering available devices, 237–238
- opening sockets, 238–241
- overview of API functionality and permissions, 236
- pairing with bonded Bluetooth devices, 238
- for smartphones, 3
- using device vibration, 241

BMP image format, 200**Bosch Sensortec, 10****Broadcast receivers**

- App Widgets, 85–87
- checking status of network connectivity, 253–255
- creating App Widgets and, 85–87
- features of, 82–83
- Little Fluffy Location Library notifications, 338–340
- push messages and, 351, 353
- SMS functionality and, 257–263
- starting service when camera button pressed, 83–85

Browsers. *see* Web browsers**Button press**

- launching activity for result using speech-to-text functionality, 42–44
- as trigger event for multiple activities, 37–38

Buttons

- aligned horizontally using LinearLayout, 116–119

- customizing for custom views, 177–182
- thumb buttons on seek bars, 141–143
- using buttons and TextView, 37–38
- using image buttons in table layout, 130–134
- using property animations for, 187–189
- using radio buttons, 130, 137–138
- using toggle buttons, 136–137
- widget, defined, 130

C

Calendar application, 191

Callback methods, 145–146. *see also* Event handlers and event listeners

CallLog, 307

Camera key, KeyEvent and, 146–147

Cameras

- customizing hardware interface, 222–226
- hardware interface, 221–226

Capacitive touchscreen technology, 8

Capella Microsystems, Inc., 10

C/C++

- building libraries using NDK, 361–370
- integrating native C code with Java Native Interface, 362–364

C/C++ Development Tooling (CDT) (Eclipse), 361–362

Check box widgets, 130, 134–137

Choices, creating list of, 44–45

CircleCI, 410

Client-side Bluetooth sockets, 238–241

Clock timers, 58–60

CMOS image sensor cameras, 3

Colors

- possible values for UI attributes, 110–111
- setting and changing text attributes, 124–127

Com.cookbook.data package

- creating personal diary, 303–306
- as separate SQLite database package, 297–300

- using separate data storage, 300–303

Compatibility pack

- adding support library to projects, 408
- Android support packages, 401–408

Connectivity manager

- determining network interfaces, 251–253
- using to access wireless networks, 241–242

Contacts

- fragments and screen displays, 191
- types of objects for, 307

Content providers

- accessing, 308, 310
- creating custom content provider, 308–312
- native Android databases as, 306–307
- optional override methods, 308
- unique URI, 308
- using loaders, 89–91

Context menus

- building of, 148–152
- examples of, 153

Continuous integration (CI) systems

- Apache Ant and, 30, 409–410
- Apache Maven and, 156, 409–410
- listing of common systems, 410
- workflow steps, 409

Coordinated Universal Time (UTC) timestamp, 317

Copyright, 16–18

Countdown timers, 60–61

CruiseControl (CI system), 410

Cupcake (Android OS 1.5, API Level 3, released 4/30/09)

- creating action bars, 156
- creating and retrieving shared preferences, 288
- features for developers, 411
- mapping the SEARCH key, 159–161

CursorLoader, advanced threading techniques, 89–91

Custom views, 177–182

D

Daemon, 381

Daemon threads, 57

Dalvik Debug Monitor Server (DDMS)

- within Android Debug Monitor, 384
- debugging through breakpoints, 380
- example of Confirm Perspective Switch dialog box, 381
- example of control panel, 379
- example of Debug perspective, 382
- installing, 13
- LogCat and, 381
- tracking memory allocation, 12
- types of debugging data, 380
- using DDMS, 378–380

Data storage methods

- content providers, 306–312
- file saving and loading, 312–314
- shared preferences, 287–297
- SQLite Database, 297–306

Databases. *see also* SQLite Database

- using AsyncTask, 91–93
- using CursorLoader, 89–91

DataStorageTester, 310–311

DDMS. *see* Dalvik Debug Monitor Server (DDMS)

Debugging

- Android SDK tools, 380–390
- Android system tools, 390–393
- Android test projects, 371–377
- creating a test project, 371–373
- Eclipse built-in tools, 377–380
- leveraging Linux tools, 390–393
- NDK-r4 and building native libraries, 363
- populating unit tests on Android, 373–376
- setting up GDB debugging, 391–393
- starting and stopping Android Debug Bridge, 380–381
- using Hierarchy Viewer, 384–386
- using lint, 388–390

- using LogCat, 381, 383–384
- using Robotium, 376–377
- using TraceView, 386–388
- when developing with USB device plugged in, 249

Design, importance of, 11

Design guidelines (Google), 11

Developers

- charging for applications, 18–19
- in-field error reports from users to, 2
- interactions with users via Google Play, 17
- managing updates and reviews, 19
- quality design, 11

Devices, running Android

- common features, 2–3
- hardware differences, 6–10
- HTC models, 3, 5
- Motorola models, 4, 5, 9
- Samsung models, 4–6
- tablets, 5–6, 7

Dialog fragments, 196–198

Diary entries, 300–306

Dimensions

- controlling width/height of UI elements, 115–119
- possible values for UI attributes, 110
- of tablet screens, 112

Directory structure

- autogenerated content, 25–26
- user-generated files, 24–25

Donut (Android OS 1.6, API Level 4, released 9/15/09)

- creating action bars, 156
- creating and retrieving shared preferences, 288
- features for developers, 411
- mapping the SEARCH key, 159–161

DPAD, KeyEvent and, 146–147

Droid Incredible, 5

Droid RAZR MAXX, 4, 5

Droid X, 5

Drop-down menus, 130, 138–140

E

Earth

gravitational field, 227–230

magnetic field, 227–230

Eclair (Android OS 2.0, API Level 5, released 10/26/09)

creating action bars, 156

creating and retrieving shared preferences, 288

features for developers, 412

introduction of separate callback method, 147

mapping the SEARCH key, 159–161

Eclipse, debugging processes

adding test case constructor, 374–375

with ADT Bundle installation, 371, 377

choosing test targets, 373, 374

creating test projects, 371–373

example of New Project wizard, 372

maintenance methods in testing, 375–376

naming test projects, 372, 373

specifying run configurations, 377–378

using DDMS, 378–380, 382

using lint, 388–390

using Robotium for executing tests, 376–377

Eclipse Integrated Development Environment (IDE)

adding Support Library, 156–157

with ADT Bundle installation, 13

Android SDK plugin for, 12

building layouts in graphical layout editor, 113–115

built-in debugging tools, 377–380

C/C++ Development Tooling (CDT), 361–362

creating projects and activities, 22–24

example of layout builder, 114

project directory structure, 25

renaming parts of application, 28–29

signing and publishing, 16

EditText

attributes, 127–128

autoresponse SMS and, 258–259

creating forms, 129–130

integrating with Twitter, 277–280

login page and, 291–293

RPCs and, 95–99

using HTTP GET and, 264–267

Emulator

ADB managing of, 381

changing rotation vector of, 397

configuring with SDK, 13–15

debugging and, 377–378, 380–381, 384, 390–391

drawbacks of, 221

as Eclipse plugin, 2

Emulator Controls, 15, 380

Hierarchy Viewer and, 115

using OpenIntents Sensor Simulator for testing applications, 395–399

Enabled location providers, 320–321

End user license agreement (EULA), 16–17, 294–297

Engine control unit (ECU), 6

EULA (end user license agreement), 16–17, 294–297

Event handlers and event listeners

building menus, 148–152

creating action bars, 154–156

defining menus in XML, 152–154

intercepting physical key press, 145–148

listening for fling gestures, 163–165

reacting to touch events, 161–163

using ActionBarSherlock, 154, 156–159

using multitouch, 165–168

using SEARCH key, 159–161

Evernote, 276

Extensible Markup Language (XML) files. *see* XML

F

Facebook

- documentation, 284
- integrating into Android applications, 284–285
- Scribe and, 276
- tutorial, 285
- virtual goods sales, 18

Facebook Android PacKage (APK), 284

Fernandez, Pablo, 276

Filenames, formatting of, 93, 109, 185, 365

FLAC audio format, 201

Flash drives, 6

Flash memory, 3

Flat file manipulation

- opening resource directories, 312–313
- using AsyncTask for asynchronous processing, 313–314

Fling gestures, 163–165

Fonts

- attributes, 110, 124–127
- dimensions attributes, 125
- setting and changing in UI elements, 124–127
- for web content, 264

Foreground services, activating, 77–80

Forms, creating and text manipulation, 129–130

Forward compatibility

- rules for maintaining, 11–12
- SDK versions and, 28

Fragments

- of activities, 35–36
- advanced user interface techniques, 191–198
- displaying multiple fragments at once, 191–196
- using bundles for serializing arguments, 36
- using dialog fragments, 196–198
- using loaders, 89–91

Frame-by-frame animation

- advanced user interface techniques, 183–189
- resource directories, 109

Free limited application versions (Google Play), 18–19

Froyo (Android OS 2.2, API Level 8, released 5/20/10)

- creating action bars, 156
- creating and retrieving shared preferences, 288
- features for developers, 412
- mapping the SEARCH key, 159–161

G

Galaxy Nexus, 4, 5

Galaxy Note, 5

Galaxy Note 2, 4, 5

Galaxy S3, 5

Galaxy Tab, 6

Gaming, 6, 315

GCM. *see* Google Cloud Messaging (GCM)

Geocoding, 324–325

Gesture Builder project, 168–171

Gestures

- advanced user interface libraries and, 168–171
- customizing, 10
- using fling gestures, 163–165

Getjar, 20

GIF image format, 200

Gifting systems, 343

Gingerbread (Android OS 2.3, API Level 9, released 12/6/10)

- accessory mode, 248
- adding notifications using Little Fluffy Location Library, 339–340
- creating action bars, 156, 158–159
- creating and retrieving shared preferences, 288
- features for developers, 412–413

mapping the SEARCH key, 159–161

Global Positioning System (GPS) navigation

- in automobiles, 6
- battery power usage, 337
- debugging and, 380
- forward compatibility and, 11
- proprietary software, 2
- satellite-based, 316
- simulation testing, 395

GNU C libraries, 2

GNU Project Debugger (GDB)

- example of output, 392
- installing, 392
- within NDK-r4, 363
- running, 392–393
- setting up, 391–393
- website address, 393

Google

- acquisition of Android, Inc., 1
- acquisition of Motorola Mobility, 5
- Android SDK website links, 12–13
- assistance to third-party developers, 2
- design guidelines, 11
- partnership with Asus, 6

Google API console, acquiring API key from, 327, 349

Google Checkout

- Google Play requirement, 16
- merchant accounts, 344
- not available in some countries, 18

Google Chrome browser, 414–415

Google Cloud Messaging (GCM), 349. *see also* Push messages, using Google Cloud Messaging library

Google Maps

- adding markers to map, 329–333
- adding to applications, 328–329
- adding views to map, 333–336
- Android API version 2, 327–328
- download and setup requirements, 325–326

- location-based services and, 322, 325–336
- maps library and permissions, 326–327
- setting up proximity alert, 336

Google Nexus 4, 4

Google Now, 159, 415

Google Play

- alternatives to, 20
- in-app billing, 343–347
- end user license agreements, 16–17
- improving visibility of application, 17
- managing reviews and updates, 19
- market differentiation of application, 18
- maxSdkVersion used as filter by, 28
- merchant accounts, 344
- monetizing applications, 18–19
- signing requirement, 16
- TalkBack download, 189

Google Play Billing Library, 344–345

Google search Representational State Transfer (REST) API

- example of search result, 268
- using HTTP GET to retrieve data, 264–268

Google TV, 177, 199

Google Wallet, 18

GPS navigation. *see* Global Positioning System (GPS) navigation

Graphic designers, 11

Graphviz dot utility, 388

Gravitational field of Earth, 227–230

Gyroscopes, 227

H

H.263 video format, 201

H.264 AVC video format, 201

Handlers (messages between threads)

- push messages and, 355
- running time-consuming initialization and, 61–63
- scheduling runnable task from main thread, 58–60

Handlers (messages between threads) (*continued*)

- using countdown timers, 60–61
- using messengers in remote processes, 99–105

Hard keyboards, 10–11

Hardware interface

- Bluetooth, 236–242
- cameras, 221–226
- getting device's rotational attitude, 227–230
- near field communication, 243–248
- sensors, 227–231
- telephony, 231–236
- universal serial bus, 248–249
- using temperature and light sensors, 230–231

HE-AACv1 (AAC+) audio format, 200

HE-AACv2 (enhanced AAC+) audio format, 200

Height, controlling dimensions of UI elements, 115–119

Hierarchy Viewer

- for debugging, 381, 384–386
- example of interface, 385
- viewing layouts with, 115, 116, 386

Holo theme, 25, 154

HOME key, KeyEvent and, 146–147

Honeycomb (Android OS 3.0, API Level 11, released 2/22/11)

- adding notifications using Little Fluffy Location Library, 339–340
- animating buttons, 187
- creating action bars, 154, 156
- creating and retrieving shared preferences, 288–289
- features for developers, 413
- mapping the SEARCH key, 159–161
- project directory structure, 24–25
- using fragments, 36

Host mode, USB devices and, 248–249

HRC One, 4

HTC, 3–5

HTC Dream (G1), 3, 9

HTC EVO 3D, 3, 5

HTC EVO 4G, 5, 9

HTC Magic, 3

HTTP GET, 264–267

HTTP POST, 267–269

Hudson (CI system), 410



Ice Cream Sandwich (Android OS 4.0, API Level 14, released 10/19/11)

- access to device owner profiles, 275
- creating and retrieving shared preferences, 288–289
- example of action bar, 156
- features for developers, 413–414
- mapping the SEARCH key, 159–161
- project directory structure, 24

IEEE standard 802.14.1, 236

Image buttons, in table layout, 130–134

Image resource directories, 109

Images

- example of scrambled image, 206
- loading and displaying for manipulation, 202–206
- multimedia techniques, 199–206
- saving bitmap picture to PNG file, 312
- supported media types (Android 4.1), 200
- using HTTP POST to retrieve web data, 267–269

ImageView, using AsyncTask, 92–93

Implicit intents for creating activity, 45–46

In-app billing (Google Play)

- adding to activities, 345–346
- boilerplate code for, 346
- completing purchase, 347
- creating listener for inventory results, 346–347
- installing, 344–345
- listing items for in-app purchase in developer console, 346–347

storing customer-identifying information, 347

versions of, 343

In-app purchases, 18–19

Inbox-style notification alert, 69

IntentService

for background tasks, 80–82

using with Result Receiver, 105

Internal pause flag, 53–55

Internet browsers. see Web browsers

Inter-process communication (IPC) protocol

AIDL interface functions, 94–95, 97

implementing remote procedure calls, 94–99

sharing threads between two applications using binders vs., 57–58

using messengers, 99–105

using ResultReceiver, 105–107

IPad, 5, 6

IPC. see Inter-process communication (IPC) protocol

iPhone, 1

IQon, 6

J

Java

capturing text entry at run-time, 129

colors of items, 111

fragments and, 193–196

OAuth module and integrating with Twitter, 276–283

programmatic layout, drawbacks of, 120–121

referencing resources, 26–28

Relative Layout rules for possible children, 120

TextView attributes, 125

Java Native Interface (JNI)

integrating native C code with, 362–364

type mapping between Java and Native, 362

Java Virtual Machine (JVM), 363

JavaScript Object Notation. see JSON (JavaScript Object Notation)

Jelly Bean (Android OS 4.1, API Level 16, released 7/9/12)

adding notifications using Little Fluffy Location Library, 340

creating and retrieving shared preferences, 288–289

example of action bar, 157

features for developers, 414–415

introduction of hard-coded SEARCH key, 159

supported media types, 200–201

Jenkins (CI system), 410

JNI. see Java Native Interface (JNI)

JPEG image format, 200

JSON (JavaScript Object Notation)

defined, 251

parsing JSON data, 271–273

using HTTP GET to retrieve web data, 264–267

website address, 264–267

JUnit, 13, 371, 375–376

JVM (Java Virtual Machine), 363

K

Keyboards

KeyEvent and, 146

and screen orientation, 34

types of, 10–11

KeyEvents, physical keys for, 145–146

Kickstarter projects, 6

Kindle Fire, 6

L

Labels for resource directories, 110

Landscape screen mode

forcing to stay constant, 34

XML layouts for, 112

Language values directories, 111

Last location, retrieving, 317–318

Latitude-longitude coordinates. *see also* Location-based services (LBS)

Little Fluffy Location Library and, 337
proximity alerts and Google Maps, 336

Layout. *see* User interface layout; Views and ViewGroups

LBS. *see* Location-based services (LBS)

Libraries

advanced user interface libraries, 168–176
Android Support Library, 156–157, 401–408
Google Cloud Messaging library, 349–360
library projects, overview of, 29–31
Little Fluffy Location Library, 337–341
Open Graphics Library for Embedded Systems (OpenGL ES), 171–176, 327, 366
third-party for integrating with Twitter, 275–276

Light sensors, 230–231

LinearLayout, 116–119

LinkedIn, 276

Lint, for debugging, 388–390

Linux OS systems

ADT Bundle for, 13
Android debugging processes and, 390–393
setting up GDB debugging, 391–393
using OpenIntents Sensor Simulator for testing applications, 396
using `top` command, 390–391

Listeners. *see* Event handlers and event listeners

Little Fluffy Location Library

adding notifications, 338
downloading, 337
example of notification, 341
location-based services and, 337–341

LiveFolder, 307

Loader API, advanced threading techniques, 89–91

Location-based services (LBS)

accuracy and power requirements, 316
application requirements, 315
listing all enabled providers, 320–321
permission to use location information, 316–317
retrieving last location, 317–318
specifying location estimation technology, 316
translating a location to address (reverse geocoding), 322–323
translating an address to location (geocoding), 324–325
updating location upon change, 318–320
using Google Maps, 322, 325–336
using Little Fluffy Location Library, 337–341

LogCat

from DDMS control panel, 379, 380
for debugging, 381, 383–384
for listening for phone states, 234
owner profiles and, 275
when developing with USB device plugged in, 249

Login page, 291–293

M

Mac OS systems

ADT Bundle for, 13
retina display, 6
using OpenIntents Sensor Simulator for testing applications, 396

Magnetic field of Earth, 227–230

Magnetometers, 9, 221, 227–230, 252

Mail animation, 184–186

Make file format, 363

Manifest files, overview of, 26–28

Margins, UI elements and, 116

Market differentiation of application, 18

MaxSdkVersion used as filter by Google Play, 28

MD5 certificate fingerprints, 326

Media button, KeyEvent and, 146

Media playback, launching secondary threads and, 52–55

MediaPlayer

- manipulating raw audio, 211
- ringtone song as secondary thread and, 52–55
- using for audio playback, 207–209
- using for video playback, 217–219

MediaStore, 217, 307

Memory

- activity lifecycle and, 32
- audio files and, 215–216
- flash drives, 6
- flash memory, 3
- foreground services and, 77
- manipulating audio and, 206, 211–213
- manipulating images and, 199, 202–204
- tracking memory allocation, 12, 390–391
- using sound resources efficiently and, 215–217

MENU key, KeyEvent and, 146–147

Menus

- building of, 148–152
- creating spinners, 113–114, 130, 138–140
- defining menus in XML, 152–154
- examples of, 153
- resource directories, 109

Messengers, in remote processes, 99–105

Micro Secure Digital (microSD) card slot, 3

Micro-electro-mechanical systems (MEMS), 227

Microprocessor unit (MPU), 3

MIDI audio format, 201

MIT License, 410

Mobile advertisement, 18–19

Monetizing applications (Google Play), 18–19

Motion events, 165

Motorola

- Android smartphones, 4, 5, 9
- app market, 20

Motorola Droid, 9

MP3 audio format, 201

MPEG-4 SP video format, 201

Multimedia techniques

- audio, 206–217
- images, 199–206
- supported media types (Android 4.1), 200–201
- video, 217–219

Multiple activities

- implementing list of choices, 44–45
- implementing remote procedure call between, 94–99
- launching activity for result using speech-to-text functionality, 42–44
- launching additional activity from event, 38–41, 42
- overview of, 36–37
- passing primitive data types between activities, 47–49
- using buttons and TextView, 37–38

Multiprocessing, App Widgets and, 10

Multitouch, 10, 165–168

N

National Semiconductor, 9

NativeActivity, 364–369

NDEF (NFC Data Exchange Format messages), 243

NDK. see Android Native Development Kit (NDK)

Near field communication (NFC)

- hardware interface, 243–248
- reading NFC tags, 243–245
- within Samsung smartphones, 5
- writing to unprotected NFC tags, 245–248

Network-based applications

- checking for connectivity, 251–253
- reacting to network state, 251–255
- receiving connectivity changes, 253–255
- social networking, 275–285

Network-based applications (*continued*)

- using SMS, 255–263
- using web content, 263–274

Nexus 7, 6

Nexus 10, 6

Nexus One, 3, 5

NFC. *see* Near field communication (NFC)

NFC Data Exchange Format (NDEF) messages, 243

O

OAuth module for Java, and integrating with Twitter, 276–283

One X+, 3

Open Graphics Library for Embedded Systems (OpenGL ES)

- for drawing 3D images, 171–176
- libraries for communication between C code and Android Framework, 366
- version 2, 327

Open Handset Alliance, 1

Open source, defined, 2

OpenIntents Sensor Simulator

- adding to application, 398–399
- downloading, 395
- Initial Settings screen, 396
- permissions, 398
- setting up, 395–397

Opera Mobile Apps Store, 20

Option menus, 148–152

Opto Semiconductor, 9

OS releases and API level. *see* **Android OS releases**, listing of

OUYA console, 6

Owner profiles of devices, 275

P

Padding, UI elements and, 116

Partial WakeLock, 74–75

Passwords

- creating private key and, 16
- NFC requirements and, 243
- shared preferences and, 287, 289–293

Pay-to-win applications, 18, 343

PCM/WAVE audio format, 201

PDU (protocol description unit), 260

Pebble watch, 6

Pending notification alerts, 65–69

Phablets, 5

Phone numbers, dialing, 235–236

Phone state listener events, 234–235

Physical key press, intercepting, 145–148

Physical keyboards, 10–11

PNG image format, 200

Portrait screen mode

- forcing to stay constant, 34
- XML layouts for, 112

Power key, **KeyEvent** and, 146–147

Preferences framework, shared preferences interface and, 288–291

Price, Kenton, 337

Pricing of applications (Google Play), 18–19

Private keys

- for OAuth, 276
- signing applications with, 16

Progress bar widget, 123, 130, 140–141

Projects. *see also* **Test projects**

- Android Asset Packaging Tool (aapt), 26
- autogenerated content, 25–26
- creating with Eclipse IDE, 22–24
- directory structure, 24–26
- user-generated files, 24–26

Protocol description unit (PDU), 260

Proximity alerts

- creating alerts without expiration time, 336
- using Google Maps and, 336

Push messages, using Google Cloud Messaging library

- adding Broadcast receiver class, 353

- adding IntentService class, 354–356
- API Access page, 351
- API service overview screen, 349–350, 350
- boilerplate code for, 359–360
- obtaining API key, 349
- permissions, 351
- preparing for setup, 349–351
- preparing the manifest, 351–353
- receiving messages, 353–356
- registering a device, 356
- sending messages, 351–353, 356–360
- sending messages with AsyncTask, 357–360
- sending text messages, 357–358
- storing API key, 358

Q

Qualcomm, Snapdragon platform, 3–4

R

- Radio button widgets, 130, 137–138**
- RAM, 3**
- Raw audio, manipulating, 211–215**
- RAZR MAXX HD, 5**
- Recording audio files, 210**
- Referencing resources**
 - Java files, 26–28
 - XML files, 26–28
- Relative Layout view, 119–120**
- Remote procedure calls. *see* RPCs (remote procedure calls)**
- Renaming parts of application, 28–29**
- Research In Motion, 1**
- Resistive touchscreen technology, 7–8**
- Resource directories**
 - language values directories, 111
 - listing of, 109
 - opening, 312

- specifying alternate resources, 111–112
- user interface layout attributes, 110–111
- REST. *see* Google search Representational State Transfer (REST) API**
- Restoring activity information, 34–35**
- ResultReceiver**
 - holds IPC binders to direct calls across multiple processes, 105–107
 - using IntentService with, 105
- Reverse geocoding, 322–323**
- Reviews by users, managing (Google Play), 19**
- RFCOMM (Bluetooth transport protocol), 238**
- Robotium**
 - downloading and tutorials, 377
 - for executing tests, 376–377

Robustness, 12

Roewe, 6

ROM. *see* Flash memory

Rotational attitude, expressing, 227–230

RPCs (remote procedure calls)

- example of output of AIDL application, 97
- implementing between two activities, 94–99
- using AIDL between processes with different user IDs, 99

RTTTL files, launching secondary threads for ringtone song, 52–55

Runnable activities

- creating, 55–56
- scheduling tasks from main thread using handlers, 58–60

S

- Saab, 6**
- Safari browser, 263**
- Samsung, 4–6**
- Satellite-based GPS, 316**
- Saving activity information, 34–35**
- Screen layout resource directories, 109**
- Screen orientation**

Screen orientation (*continued*)

- forcing to stay constant, 34
- keyboard slide-out events and, 34
- XML layouts for, 112

Screen resolution, 111**Screens**

- AMOLED displays, 7
- light sensors and, 230–231
- specifications of, 8
- of tablets, 112
- TFT LCDs, 7
- touchscreens, 7–8, 10

Scribe, 276**SDK.** *see* **Software Development Kit (SDK)****SDRAM/RAM** (synchronous dynamic random access memory), 3**SEARCH** key

- KeyEvent and, 146–148
- using with event handlers and event listeners, 159–161

SearchRecentSuggestions, 307**Secondary threads**

- launching ringtone song, 52–55
- updating layouts from separate thread, 121–124
- when accessing web data, 268

Seek bar widgets, 130, 141–143**Self-contained services**

- adding WakeLocks, 74
- creating, 70–74

Sensors. *see also* **OpenIntents Sensor Simulator**

- light sensors, 230–231
- SDK supported sensors, listing, 227
- smartphones as sensor hubs, 8–10
- temperature sensors, 230–231
- types of, 9

Server-side Bluetooth sockets, 238–241**Services**

- adding WakeLocks to self-contained service, 74–77
- creating self-contained, 70–74

defined, 69

lifecycle flowchart, 71

scenarios of, 70

using an IntentService, 80–82

using foreground services, 77–80

Settings

as content provider native database, 307

forward compatibility and, 11, 28

Hierarchy Viewer and, 115

shared preferences interface and, 287–293

Shanghai Automotive Industry Corporation, 6**Shared preferences**

adding an EULA, 294–297

changing the UI based on stored data, 290–293

creating and retrieving, 288

as data storage method, 287–297

login page, 290–293

using the preferences framework, 288–290, 291

Short message service (SMS)

autoresponse SMS based on received SMS, 257–263

located in android.telephony package, 257–263

networked-based applications and, 255–263

push messages and, 357–358

retrieving protocol description unit, 260

setting messages to 140 characters or less, 257, 275

Single task mode, forcing, 31–34**SlideMe, 20****Smartphones.** *see also* **Telephony**

models of, 3, 4

sensors and, 8–10, 227–231

SMS. *see* **Short message service (SMS)****Snapdragon platform, 3****Social networking**

integrating with Facebook, 18, 276, 284–285

- integrating with Twitter, 275–283
- networked-based applications and, 275–285
- reading owner profile of devices, 275

Soft keyboards, 10–11, 128–129

Software Development Kit (SDK)

- Android Debug Bridge (ADB), 15–16
- configuring emulators, 14–15
- debugging tools, 14–16, 380–390
- downloading support library, 408
- installing, 12–13
- OS releases and API level, 14, 411–415
- release 14 includes library projects, 29
- signing and publishing, 16
- supported sensors in, 227
- upgrading, 12–13

Spacing, UI elements and, 116–119

Speech-to-text functionality, 42–44

Spelling corrections, 129

Spinner widgets, 113–114, 130, 138–140

SQLite Database

- creating personal diaries, 303–306
- creating separate database packages, 297–300
- ListView of diary entries, 307
- using separate database packages, 300–303

ST Microelectronics, 9

Standard graphical widgets. *see also* App Widgets

- creating spinners, 130, 138–140
- using check boxes, 134–136
- using image buttons in table layout, 130–134
- using progress bars, 123, 130, 140–141
- using radio buttons, 137–138
- using seek bars, 130, 141–143
- using toggle buttons, 136–137

Standby, adding WakeLocks, 74

Status bar pending notification alerts, 65–69

Storage. *see* Data storage methods

Strings, 110

Submenus

- building of, 148–152
- examples of, 153

Support packages

- android.support.v4.accessibilityservice package, 401
- android.support.v4.app package, 402–403
- android.support.v4.content package, 404
- android.support.v4.content.pm package, 404
- android.support.v4.database package, 404
- android.support.v4.net package, 405
- android.support.v4.os package, 405
- android.support.v4.util package, 405
- android.support.v4.view package, 405–407
- android.support.v4.view.accessibility package, 407
- android.support.v4.widget package, 408

Surface acoustic touchscreen technology, 8

Synchronous dynamic random access memory (SDRAM/RAM), 3

SyncStateContract, 307

T

Table Layout, using image buttons in, 130–134

Tablets

- Android, listing of, 7
- fragments and screen displays, 191
- overview of, 5–6
- screen dimensions for, 112
- using fragments, 35

TalkBack

- downloading, 189
- voice synthesis service, 189–190

Telephony

- dialing phone numbers, 235–236
- hardware interface, 231–236
- listening for phone states, 234–235

Telephony (*continued*)

- permissions, 234
- using telephony manager, 231–233

Telephony manager, 231–233**Temperature sensors, 230–231****Test projects**

- creating using Eclipse, 371–373
- debugging and, 371–377

Text attributes, 124–127**Text entry**

- auto-capitalization, 129
- spelling correction, 129
- for user input, 127–129
- using soft keyboards, 128–129
- word suggestions, 129

Text manipulation, of UI elements

- creating forms, 129–130
- providing text entry, 127–129
- setting and changing text attributes, 124–127

Text messages. *see* Short message service (SMS)**TextView**

- attributes, 125
- showing results of multiple activities, 37–38

Thin-film transistor (TFT) LCDs, 7**Third-party application stores, 18, 20****Thread priorities, setting of, 56–57****Threading techniques, advanced**

- AxyncTask, 91–93
- implementing remote procedure call, 94–99
- inter-process communication (IPC) protocol, 94–107
- loaders, 89–91
- using CursorLoader, 89–91
- using messengers, 99–105
- using ResultReceiver, 105–107

Threads

- canceling, 57

- creating runnable activities, 55–56
- handlers, 58–63
- launching secondary threads, 52–55
- overview of, 51
- setting thread priorities, 56–57
- sharing between two applications, 57–58
- updating layouts from separate thread, 121–124

3-bit TNF field, 243**3D images, 171–176****Three-axis accelerometers, 9–10, 227–230****Three-axis magnetometers, 9, 227–230****Thumb buttons, 141–143****Time-consuming initialization, using handlers and, 61–63****Toggle button widgets, 136–137****Top command for debugging, 390–391****Touch events, 10, 161–163****Touchscreen technology, 7–8, 10****TraceView**

- example of analysis screen, 388
- for optimizing performance, 381, 386–388
- specifying factorial method, 386–387
- trace log files and, 386–388

Trackball, KeyEvent and, 146**TV screens, using fragments, 35****Tween animation**

- advanced user interface techniques, 183–189
- resource directories, 109

Twitter

- features of, 275
- integrating into Android applications, 275–283
- registering applications with, 276
- Scribe and, 276
- third-party libraries for integrating with, 275–276

Twitter4J, 276, 283

U

Uniform resource identifier (URI)

- implicit intents and, 45–46
- NFC tags and, 243
- requirement for content providers, 308–309

Universal serial bus (USB) devices

- accessory mode, 248–249
- ADB managing of, 248–249, 381
- Android devices as emulators and, 14
- hardware interface, 248–249

Updates, managing (Google Play), 19**URI. *see* Uniform resource identifier (URI)****User input methods, 7–8, 127–129****User interface events**

- advanced user interface libraries, 168–176
- building menus, 148–152
- creating action bars, 154–156
- defining menus in XML, 152–154
- event handlers and event listeners, 145–164
- intercepting physical key press, 145–148
- listening for fling gestures, 163–165
- reacting to touch events, 161–163
- using ActionBarSherlock, 156–159
- using multitouch, 165–168
- using SEARCH key, 159–161

User interface layout

- general attributes, 110–111
- resource directories, 109–112
- text manipulation, 124–130
- views and ViewGroups, 112–124
- widgets. *see* Standard graphical widgets

User interface libraries, advanced

- drawing 3D images, 171–176
- using gestures, 168–171

User interface techniques, advanced

- accessing accessibility features, 189–191
- animation, 183–189

custom views, 177–182

fragments, 191–198

UserDictionary, 307–308**Username objects, 291–293**

UUID (universally unique identifier), opening
Bluetooth sockets and, 239–240

V

Vibration, in Bluetooth devices, 241**Video**

- multimedia techniques, 217–219
- playback using MediaPlayer, 219
- supported media types (Android 4.1), 201
- using HTTP POST to retrieve web data, 267–269
- using VideoView, 217–219

VideoView, 217–219**Views and ViewGroups**

- building layouts in Eclipse editor, 113–115
- controlling width/height of UI elements, 115–119
- custom views, 177–182
- declaring programmatic layout, 120–121
- example of horizontally placed widgets, 113
- setting Relative Layout and layout ID, 119–120
- updating layouts from separate thread, 121–124

Vimeo, 276**Virtual goods sales, 18****Visibility of applications (Google Play), 17****Volume key, KeyEvent and, 146–147****Vorbis audio format, 201****VP8 video format, 201**

W

WakeLocks

- adding to self-contained services, 74–77

WakeLocks (*continued*)

- comparison of types, 75
- push messages and, 351

Web browsers

- customizing, 263–264
- Google Chrome browser, 414–415
- Google Maps and, 325
- incognito mode, 413
- native Android databases as content provider, 306
- Safari browser, 263

Web content

- customizing web browsers, 263–264
- networked-based applications and, 263–274
- parsing JSON data, 271–273
- parsing XML data, 273–274
- using an HTTP GET to retrieve web data, 264–267
- using HTTP POST to retrieve data, 267–269
- using WebViews, 269–271

WebKit, 263–264**WEBP image format, 200****WebViews, 269–271****What-you-see-is-what-you-get (WYSIWYG) user interface, 377****Widgets. *see* App Widgets; Standard graphical widgets****Width, controlling dimensions of UI elements, 115–119****Wi-Fi (802.11)**

- cell tower identification, 316
- checking for device connectivity to, 251–253, 255
- debugging and, 249
- smartphones and, 3
- tablets and, 5

WiMAX (802.16e-2005), 5**Windows OS systems**

- ADT Bundle for, 13
- integrating with Facebook, 284
- SDK drivers for, 14
- using lint tool with, 388–390

Wireless networks, 241–242**Word suggestions, text entry and, 129****Wrist watches, with Android systems, 6**

X

X Windows, 2**XML**

- arbitrary filenames, 109
- colors of items, 111
- creating animation with, 187–189
- defining layouts for screen types, 112
- defining menus, 152–154
- EditText attributes, 128
- Google Maps and, 327
- labels and text of items, 110
- measurements and dimensions of items, 110
- parsing XML data, 273–274
- project user-generated files, 24–25
- referencing resources, 26–28
- Relative Layout rules for possible children, 120
- with resource descriptors, 109
- resource directories, 109
- shared preferences interface and, 287
- TextView attributes, 125
- using HTTP GET to retrieve web data, 264–267

Y

Yamamoto, Yusuke, 276