

Carmen Delessio  
Lauren Darcey  
Shane Conder

Third Edition

Sams **Teach Yourself**

# Android™ Application Development

in **24**  
Hours

SAMS

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



**Praise for**  
**Sams Teach Yourself**  
**Android™ Application Development**  
**in 24 Hours, Third Edition**

“True, Android has its own spin on the Java programming language, particularly if you come from a traditional mobile Java background. Don’t be flummoxed! Use this book to quickly get over those initial hurdles so you can see Android’s inner beauty. And Android is beautiful when presented in this effective and confidence-building format.”

—**David Oliver**, President, Oliver+Coady, Inc.

“*Sams Teach Yourself Android Application Development in 24 Hours* begins in Hour 1 by providing a comprehensive overview of Android and the Eclipse development environment. In the subsequent hours, the authors provide in-depth coverage of a wide array of topics, from explanation of Android basics to advanced topics such as using the camera, location services, and connecting to the Flickr API. The concise explanations of Android concepts and focused project instruction make this book perfect for novice and intermediate Android developers.”

—**Valerie Shipbaugh**, Assistant Professor, Stark State College

“Let Android Sherpas Carmen Delessio, Lauren Darcey, and Shane Conder be your guides. They will take you on an efficient journey through Android territory, whether you are just starting out or a pro looking for a second set of eyes. Just the right amount of time is spent on the important landmarks, without getting bogged down in the details only a local would need to know.”

—**Jason Van Anden**, Developer of BubbleBeats and the I’m Getting Arrested Android apps

*This page intentionally left blank*

Carmen Delessio  
Lauren Darcey  
Shane Conder

Sams **Teach Yourself**

# Android™

## Application Development

in **24**  
**Hours**

Third Edition

**SAMS**

800 East 96th Street, Indianapolis, Indiana, 46240 USA

# Sams Teach Yourself Android™ Application Development in 24 Hours, Third Edition

Copyright © 2014 by Carmen Delessio, Lauren Darcey, and Shane Conder

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Several images in this book use scenes from the online movie Big Buck Bunny to illustrate the use of online video and using a VideoView control. This movie and related material is distributed under a Creative Commons license. For more information on the movie, go to <http://www.bigbuckbunny.org/>.

Blender Foundation | [www.blender.org](http://www.blender.org)

© Copyright 2008, Blender Foundation / [www.bigbuckbunny.org](http://www.bigbuckbunny.org)

Some images in this book are reproduced or are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

See <https://developers.google.com/readme/policies>.

Screenshots of Google Products follow these guidelines:

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

The following are registered trademarks of Google:

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

<https://www.facebookbrand.com/top-questions>

Facebook™ and Facebook Platform™ are registered trademarks of Facebook.

Flickr™ and Flickr API™ are registered trademarks of Yahoo!.

Flickr API Screenshots follow the fair use guidelines found here:

<http://pressroom.yahoo.net/pr/ycorp/permissions.aspx>

No Flickr end user images appear in this book.

ISBN-13: 978-0-672-33444-3

ISBN-10: 0-672-33444-5

Library of Congress Control Number: 2013944817

Printed in the United States of America

Second Printing: February 2014

## Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

## Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

## Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

[corpsales@pearsontechgroup.com](mailto:corpsales@pearsontechgroup.com)

For sales outside of the U.S., please contact

International Sales

[international@pearsoned.com](mailto:international@pearsoned.com)

### Editor-in-Chief

Mark Taub

### Executive Editor

Laura Lewin

### Development Editor

Sheri Cain

### Managing Editor

Kristy Hart

### Project Editor

Andy Beaster

### Copy Editor

Paula Lowell

### Indexer

Tim Wright

### Proofreader

Sarah Kearns

### Technical Editors

Sebastian Delmont

Valerie Shipbaugh

### Publishing Coordinator

Olivia Basegio

### Interior Designer

Gary Adair

### Cover Designer

Mark Shirar

### Composer

Gloria Schurick

# Contents at a Glance

Preface .....xiv

## Part I: Getting Started

**HOURL 1** Getting Started: Creating a Simple App ..... 3  
    **2** Understanding an Android Activity ..... 19  
    **3** Exploring an Android Project and Resources ..... 43  
    **4** Not Just Smartphones: Supporting Tablets, TVs, and More ..... 59

## Part II: User Interface

**HOURL 5** Using Layouts ..... 79  
    **6** Working with Basic UI Controls ..... 95  
    **7** ActionBar and Menu Navigation ..... 113  
    **8** Activities and Fragments ..... 127  
    **9** Alert! Working with Dialogs ..... 141  
    **10** Lists, Grids, Galleries, and Flippers ..... 157  
    **11** App Setting: Managing Preferences ..... 171

## Part III: Data Access and Use

**HOURL 12** Accessing the Cloud: Working with a Remote API ..... 189  
    **13** Using SQLite and File Storage ..... 205  
    **14** Creating a Content Provider ..... 227  
    **15** Loaders, CursorLoaders, and CustomAdapters ..... 245  
    **16** Developing a Complete App ..... 259

## Part IV: Special Topics

**HOURL 17** Contacts and Calendar: Accessing Device Data ..... 283  
    **18** Where Are We? Working with Location-Based Services ..... 299  
    **19** Bonjour, World! Localizing Your Apps ..... 317  
    **20** Say Cheese! Working with Cameras ..... 329

**21** Media Basics: Images, Audio, and Video..... 347  
**22** Using the Facebook SDK..... 365

**Part V: Wrapping Up**

**HOUR 23** Pro Tips, Finishing Touches, and Next Steps ..... 383  
**24** Publishing Your Apps ..... 399  
Index ..... 413

# Table of Contents

<b>Preface</b>	<b>xiv</b>
New in the Third Edition.....	xiv
Who This Book Is For .....	xv
How This Book Is Organized.....	xv
<b>Part I: Getting Started</b>	
<b>HOOR 1: Getting Started: Creating a Simple App</b>	<b>3</b>
Setting Up Your Development Environment .....	3
Instantly Creating a Simple App .....	6
Understanding the Java Code and XML Layout .....	10
Running the App.....	11
Personalizing the App .....	13
Summary .....	17
Q&A.....	17
Workshop.....	18
Exercise .....	18
<b>HOOR 2: Understanding an Android Activity</b>	<b>19</b>
Understanding an Activity.....	19
Starting an Activity .....	20
Passing Information Between Activities .....	28
Understanding Intents.....	34
Understanding the Activity Lifecycle .....	38
Summary .....	41
Q&A .....	41
Workshop.....	42
Exercises.....	42
<b>HOOR 3: Exploring an Android Project and Resources</b>	<b>43</b>
Exploring the Android Project Files .....	43
Understanding Common Resources.....	47



Summary .....	57
Q&A .....	57
Workshop.....	57
Exercises.....	58

#### **HOOR 4: Not Just Smartphones: Supporting Tablets, TVs, and More** **59**

A Brief History of Android.....	59
Handling Device Display and Orientation.....	60
Device Features .....	67
Platform Versions and the Compatibility Package .....	68
Launching Apps on a Device.....	72
Summary .....	74
Q&A .....	74
Workshop.....	75
Exercises.....	75

### **Part II: User Interface**

#### **HOOR 5: Using Layouts** **79**

Getting Started with Layouts .....	79
More Layout Basics with LinearLayout .....	84
Laying Out Child Views .....	86
FrameLayout.....	89
RelativeLayout.....	89
Summary .....	91
Q&A .....	92
Workshop.....	92
Exercises.....	93

#### **HOOR 6: Working with Basic UI Controls** **95**

Setting Up the Demo App .....	95
Using Input Controls.....	98
Using Controls with Adapters .....	104
ProgressBars and SeekBars .....	107
ImageViews.....	110
Summary .....	111

Q&A .....	111
Workshop.....	112
Exercise .....	112
<b>HOOR 7: ActionBar and Menu Navigation</b>	<b>113</b>
Understanding the Options Menu .....	113
Using the Action Bar .....	120
Strategies for Using the ActionBar and Menus .....	124
Summary .....	125
Q&A .....	125
Workshop.....	126
Exercises.....	126
<b>HOOR 8: Activities and Fragments</b>	<b>127</b>
Using Fragments Across UIs .....	127
Creating and Displaying Fragments.....	128
Using Fragments for Navigation.....	132
Fragment and Activity Interaction .....	136
Summary .....	139
Q&A .....	139
Workshop.....	139
Exercises.....	140
<b>HOOR 9: Alert! Working with Dialogs</b>	<b>141</b>
Understanding a Dialog Fragment.....	141
Dialogs for Picking Date and Time.....	145
Using Alert Dialogs.....	151
Summary .....	155
Q&A .....	156
Workshop.....	156
Exercise .....	156
<b>HOOR 10: Lists, Grids, Galleries, and Flippers</b>	<b>157</b>
ListFragments.....	157
Grids and Galleries.....	162
Using an AdapterViewFlipper .....	166
Options for Paging Controls .....	167

Summary .....	168
Q&A .....	169
Workshop.....	169
Exercise .....	169

### **HOOR 11: App Setting: Managing Preferences** **171**

Using SharedPreferences.....	171
Setting User Preferences.....	174
Summary .....	185
Q&A .....	185
Workshop.....	186
Exercise .....	186

## **Part III: Data Access and Use**

### **HOOR 12: Accessing the Cloud: Working with a Remote API** **189**

Fetching Remote Data.....	190
Using and Parsing JSON-Formatted Data .....	194
Putting the Pieces Together.....	197
Checking Connectivity .....	202
Summary .....	203
Q&A .....	203
Workshop.....	203
Exercise .....	204

### **HOOR 13: Using SQLite and File Storage** **205**

Organizing a Database with Tables.....	205
Managing Data with SQLiteOpenHelper .....	206
Adding, Updating, and Deleting Data .....	209
Querying Data and Using Cursors.....	212
Using a Database in the App.....	215
Saving an Image File .....	218
Summary .....	224
Q&A .....	224
Workshop.....	225
Exercises.....	225

<b>HOURL 14: Creating a Content Provider</b>	<b>227</b>
Using a URI for Data Retrieval .....	227
Building a Content Provider .....	228
Using FlickrPhotoProvider in the App.....	236
Requesting a File from a Content Provider.....	237
Summary .....	242
Q&A .....	242
Workshop.....	243
Exercises.....	243
<b>HOURL 15: Loaders, CursorLoaders, and CustomAdapters</b>	<b>245</b>
How Loaders Work .....	245
Loader Classes .....	246
Understanding Loader States .....	246
Creating Custom Adapters.....	253
Summary .....	257
Q&A .....	257
Workshop.....	258
Exercise .....	258
<b>HOURL 16: Developing a Complete App</b>	<b>259</b>
Determining App Functionality.....	259
Developing the App .....	263
Summary .....	278
Q&A .....	278
Workshop.....	278
Exercise .....	279
<b>Part IV: Special Topics</b>	
<b>HOURL 17: Contacts and Calendar: Accessing Device Data</b>	<b>283</b>
All About the Calendar.....	283
Understanding Contacts.....	294
Summary .....	297
Q&A .....	297

Workshop.....	298
Exercise .....	298
<b>HOUR 18: Where Are We? Working with Location-Based Services</b>	<b>299</b>
Determining Location .....	299
Using Geocoding Services.....	306
Using the Geo Intent .....	309
Additional Location Features.....	311
Using Google Play Services.....	312
Summary .....	315
Q&A .....	316
Workshop.....	316
Exercise .....	316
<b>HOUR 19: Bonjour, World! Localizing Your Apps</b>	<b>317</b>
General Internationalization Principles .....	317
Working with Localization with Android.....	318
Handling Locales with Android.....	319
Using Applications to Handle Locales.....	320
Android Internationalization Strategies.....	323
Using Localization Utilities .....	326
Summary .....	327
Q&A .....	327
Workshop.....	327
Exercises.....	328
<b>HOUR 20: Say Cheese! Working with Cameras</b>	<b>329</b>
Capturing Media .....	329
Using Intents to Take Photos and Videos .....	332
Developing a Camera App .....	338
Summary .....	345
Q&A .....	346
Workshop.....	346
Exercise .....	346

<b>HOUR 21: Media Basics: Images, Audio, and Video</b>	<b>347</b>
Examining the ImageView Control .....	347
Bitmaps and Canvas .....	353
Using VideoViews .....	357
Playing Audio with MediaPlayer .....	361
Exploring More Media Options.....	362
Summary .....	362
Q&A .....	362
Workshop.....	363
Exercise .....	363
<b>HOUR 22: Using the Facebook SDK</b>	<b>365</b>
About Facebook .....	365
Setting Up for Facebook Development .....	366
Using the Facebook SDK in a Project.....	370
Developing a Facebook Photo Upload App.....	373
Facebook SDK Features.....	378
Creating Libraries for Your Own Projects .....	379
Summary .....	379
Q&A .....	380
Workshop.....	380
Exercise .....	380
<b>Part V: Wrapping Up</b>	
<b>HOUR 23: Pro Tips, Finishing Touches, and Next Steps</b>	<b>383</b>
Responsive Apps: Using IntentService.....	383
Adding Animation .....	388
Using Open Source .....	391
Digging Deeper into Android.....	392
Summary .....	397
Q&A .....	397
Workshop.....	397
Exercise .....	398

<b>Hour 24: Publishing Your Apps</b>	<b>399</b>
Preparing for Release.....	399
Publishing Your App .....	405
Monetizing Your App .....	409
Summary .....	410
Q&A .....	410
Workshop.....	410
Exercise .....	411
<b>Index</b>	<b>413</b>

# Preface

*What I wish I knew when I started Android development.*

Android has become a leading platform for smartphones, tablets, and other devices. The size of the potential global audience for your apps is justification to learn Android. The goal of this book is to introduce the Android platform and empower you to create professional grade apps.

In 24 hours of focused material, you will learn the basics of Android development and move on to specific topics like working with data in the cloud, handling bitmaps and videos in an app, and using the Facebook Android SDK.

In the early days of Android, author Carmen Delessio worked on a major Android project for a large media company. The app launched and was a success. But, it really was not built in an Android way. Having built many Android apps since then, the material in this book is largely guided by the idea of including “What I wish I knew back then.”

The material in this book covers common topics in professional Android development. The book is not intended to be an encyclopedia of all things Android. Plenty of Android resources are available and the documentation on the Android developer site has never been better. This book will start you on the path to developing professional Android apps and can be used as a guide to the additional material.

## New in the Third Edition

There are three major changes from the second edition to the third edition of this book.

New features in Android are covered. The updates include significant coverage of fragments, the action bar, and the Android support package. This edition covers new open-source projects that help in Android development, differences in Android versions, and how to use the support package to develop for multiple versions of Android.

More topics are covered in the third edition. Topics that are covered in detail in this edition include using SQLite, creating content providers, and using the Facebook SDK.

The final change is structural. In the second edition of this book, a single app was developed and expanded on in each hour. As new features were added to the app, that feature was



described in the book. This approach helped many Android developers get their start and was a way to build a real app along with the authors.

This third edition takes a topic-based approach. For each topic covered, an example project is developed. Some hours include multiple projects. The goal of these examples is to clearly illustrate the topic being covered, which might mean that the examples, particularly in the early hours, do not simulate real-world apps. In Hours 12–16, which cover cloud data and the Flickr API, the chapters are related to and build on each other. Hour 12 begins with retrieving Flickr data, and Hour 16 is a complete app that uses the Flickr data and displays images. The later hours in the book cover single topics such as using the camera or using the Facebook SDK. For each of those hours, an example project is developed.

## Who This Book Is For

The examples in this book are created so that someone with programming knowledge can understand them, but Android apps are developed in Java. The book will be much more valuable and useful if you are familiar with Java.

If you are a Java programmer with an interest in Android development, this book will introduce you to Android and get you on track for professional Android development.

If you have started Android development, but have not proceeded past the basic examples, then this book is for you. It covers topics such as downloading data, using a database, and creating content providers. The book can take you from the basics to real development in a series of understandable steps.

## How This Book Is Organized

Hours 1–4 are a hands-on “getting started” section for Android development. In these hours, you will set up your development environment, create and run an example app, and go over the details of how to set up and organize an Android project. Android resources are used to customize apps for different devices and languages and they are introduced in this section.

Hours 5–11 cover all aspects of developing user interfaces on Android, including layouts for basic controls as well as advanced controls such as ViewFlippers.

In Hours 12–16, you learn to develop an app using the Flickr API. The Flickr API includes an option to get recently uploaded photos. In Hour 12, the data for the API is received and parsed. Over several hours, you learn to develop an app that parses the Flickr data, stores it locally, and uses it to display the associated images.

Hours 17–22 cover specific topics: using contact and calendar data, location-based services, internationalization, working with cameras, using media, and developing a Facebook Android app.

Hour 23 includes tips and tricks for creating a responsive app and touches on Android topics that you might want to pursue further.

Hour 24 shows how to package and publish your app.

# About the Authors

**Carmen Delessio** is an experienced application developer who has worked as a developer, technical architect, and CTO in large and small organizations.

Carmen developed the award-winning “BFF Photo” Android app, which won the Sprint App Challenge contest in the Social Networking category.

Carmen began his online development career at Prodigy, where he worked on early Internet applications, shopping apps, and fantasy baseball.

He has written for Mashable and AndroidGuys and is the author of *Sams Teach Yourself Google TV App Development in 24 Hours*.

He is a graduate of Manhattanville College and lives in Pound Ridge, New York, with his wife, Amy, and daughter, Natalie.

**Lauren Darcey** is a multi-published Android author with several Pearson books to her credit, along with earlier editions of this book. Lauren is technical leader of Mamlambo, Inc., a firm specializing in mobile development and consulting with Android, iOS, Blackberry, and other mobile platforms. She has more than two decades of software development experience under her belt and is a recognized authority in enterprise architecture and commercial-grade mobile development.

**Shane Conder** is also a multi-published Android author with several Pearson books to his credit, along with earlier editions of this book. Shane has extensive development experience and has focused his attention on mobile and embedded development for the past two decades. He has designed and developed many commercial apps for Android, iOS, BREW, Blackberry, J2ME, Palm, and Windows Mobile, some of which have been installed on millions of phones worldwide.

# Dedication

*For Amy and Natalie.*

—*Carmen Delessio*

# Acknowledgments

This book would not exist without the help and guidance of the team at Pearson (Sams Publishing). Thanks to Laura Lewin for constant encouragement and Olivia Basegio for her incredible work on the project.

Technical editors are an important part of every book. That is particularly true in this case. Sebastian Delmont is a Ruby on Rails expert who also happens to be an Android guru. Valerie Shipbaugh did her technical review by placing herself in the role of a reader who was new to Android. The feedback and guidance from Sebastian and Valerie make this a much better book.

# We Want to Hear from You!

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

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

*Please note that we cannot help you with technical problems related to the topic of this book.*

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

Email: [consumer@samspublishing.com](mailto:consumer@samspublishing.com)

Mail: Sams Publishing  
ATTN: Reader Feedback  
800 East 96th Street  
Indianapolis, IN 46240 USA

## Reader Services

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

# HOUR 21

## Media Basics: Images, Audio, and Video

---

### What You'll Learn in This Hour:

- ▶ Examining the `ImageView` control
- ▶ Using bitmaps and canvas
- ▶ Playing audio
- ▶ Playing video
- ▶ Checking out more media options

Images and media can play an important role in creating an exceptional Android app. This chapter takes a closer look at the details of handling images and bitmaps including creating bitmaps, using drawing commands, and handling very large images. It also covers audio, video, and other media options.

### Examining the `ImageView` Control

Hour 6 showed how to use a simple `ImageView`. In that case, you saw how to display a drawable resource in the `ImageView`. An `ImageView` can display any drawable image. The source of the image can be a resource, a drawable, or a bitmap.

The source code for basic `ImageView` examples are in the accompanying `Hour21ImageView` project.

The four projects that contain the source code for this hour are the following:

- ▶ `Hour21ImageView`
- ▶ `Hour21LargeImage`
- ▶ `Hour21VideoView`
- ▶ `Hour21Audio`

## Displaying an Image

Four methods are available for setting an image in an `ImageView`. They differ by how the image passed is defined. The image can be a bitmap, drawable, URI, or resource id. The methods are as follows:

- ▶ `setImageDrawable()`: Set a drawable as the content of the `ImageView`.
- ▶ `setImageBitmap()`: Set a bitmap as the content of the `ImageView`.
- ▶ `setImageResource()`: Use a resource id to set the content of the `ImageView`.
- ▶ `setImageUri()`: Use a URI to set the content of the `ImageView`.

To set an `ImageView` to an image resource defined by `R.drawable.mainImage`, you would use the following:

```
ImageView mainImage = (ImageView) findViewById(R.id.imageView1);
mainImage.setImageResource(R.drawable.mainImage)
```

In this hour, you take a closer look at using bitmaps with `ImageViews`.

In Hour 3, you used a `ShapeDrawable` that had been defined as a resource. If you were working with a `ShapeDrawable` in code, you would use the `setImageDrawable()` method.

To populate a `Drawable` object from a resource, use the `getResources.getDrawable()` method:

```
Drawable myDrawable = getResources().getDrawable(R.drawable.ic_launcher);
```

You'll populate an `ImageView` using a resource id to explore some of the available features in an `ImageView`.

## Using ScaleTypes in ImageView

`ImageViews` include a `ScaleType` property. The `ScaleType` defines how an image displays within the `ImageView`. Using `ScaleType`, you can have an image fill the entire `ImageView`, be centered in the `ImageView`, or be cropped and centered in the `ImageView`.

The options for `ScaleType` are defined in `ImageView.ScaleType`. For example, `ImageView.ScaleType.CENTER` refers to a scale type in which the image is centered in the `ImageView`.

All scale types except for `FIT_XY` and `FIT_MATRIX` maintain the aspect ratio.

The complete set of `ScaleTypes` includes the following:

- ▶ `ImageView.ScaleType.CENTER`: Center the image with no scaling. The image dimensions are unchanged.
- ▶ `ImageView.ScaleType.CENTER_CROP`: Scales the image and keeps the aspect ratio until either the width or height of the image is the same as the width or height of the `ImageView`. For a small image, this will have the effect of enlarging the entire image. For a large image, this will have the effect of showing the center of the image.
- ▶ `ImageView.ScaleType.CENTER_INSIDE`: Scale the image and maintain aspect ratio. The width and height of the image fit within the `ImageView`.
- ▶ `ImageView.ScaleType.FIT_CENTER`: Fit the image in the center of the `ImageView`.
- ▶ `ImageView.ScaleType.FIT_START`: Fit the image in the left and top edge of the `ImageView`.
- ▶ `ImageView.ScaleType.FIT_END`: Fit the image in the right and bottom edge of the `ImageView`.
- ▶ `ImageView.ScaleType.FIT_XY`: Fit into the length and width of the `ImageView`. Does not maintain the aspect ratio.
- ▶ `ImageView.ScaleType.MATRIX`: Scale using a matrix. This allows for more complex translations of the image and is described later this hour.

The app created in the `Hour21ImageView` project illustrates the effects of setting different values for `ScaleType`. The layout file for this app includes an `ImageView` that fills the layout and a set of radio buttons for setting the `ScaleType`. `MainActivity.java` is set up in a simple way. With the `ImageView` and `RadioButtons` defined, when a button is clicked, the `ScaleType` value is set in the `ImageView`. Listing 21.1 shows the `onCheckedChangeListener()` method for the `RadioButton` named `centerCrop`. If this `RadioButton` is checked, the `ScaleType` for `ImageView` is updated.

### LISTING 21.1 Changing ScaleType Programmatically

---

```

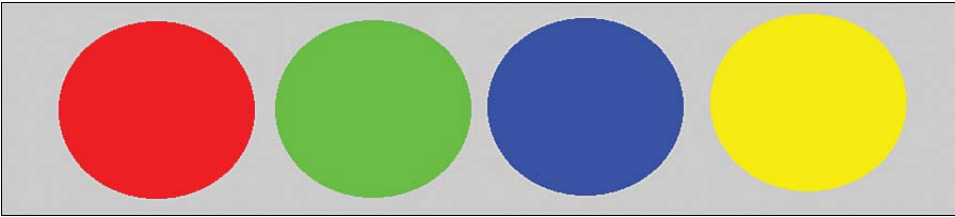
1: centerCrop.setOnCheckedChangeListener(new OnCheckedChangeListener() {
2:     @Override
3:     public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
4:         if (isChecked){
5:             imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
6:         }
7:     }
8: });

```

---



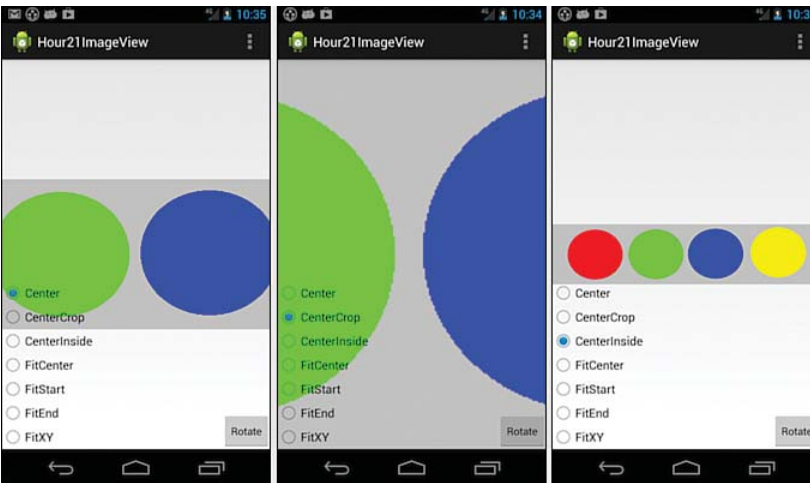
The Hour21Image app uses the image shown in Figure 21.1 as the image for the `ImageView`. The image is 900 pixels wide and 200 pixels high.



**FIGURE 21.1**  
Base image for showing `ScaleType` (scaletest.png)

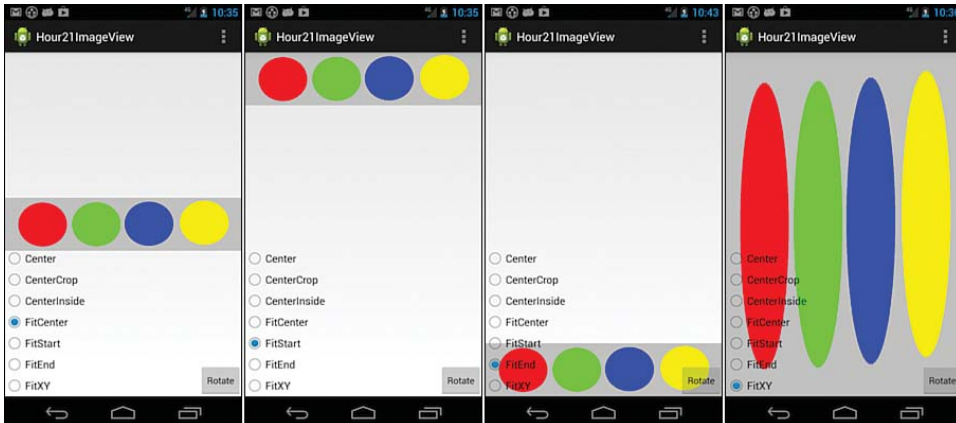
By using this simple image with four circles of different colors, you can easily see the effect of the changing `ScaleType`.

Figure 21.2 shows the base image using the `ScaleTypes` `CENTER`, `CENTER_CROP`, and `CENTER_INSIDE`. Using `CENTER` shows the image in actual size. Because the size of the image is larger than the `ImageView`, the green and blue circles in the center display. `CENTER_CROP` shows half of the green and half of the blue circles. The height of the image fills the `ImageView`. `CENTER_INSIDE` shows the entire image centered in the `ImageView`.



**FIGURE 21.2**  
`ScaleTypes` `CENTER`, `CENTER_CROP`, and `CENTER_INSIDE`

Figure 21.3 shows the base image using the `ScaleTypes` `FIT_CENTER`, `FIT_START`, `FIT_END`, and `FIT_XY`. The aspect ratio is maintained in the first three options, but when you use `FIT_XY`, the image fills the `ImageView` and “stretches” the image to fit.



**FIGURE 21.3**  
ScaleTypes `FIT_CENTER`, `FIT_START`, `FIT_END`, and `FIT_XY`

## Rotating an Image with Matrix

In graphics programming, a matrix is used to transform an image. Simple transformations include scaling, translating, or rotating an image. Android includes a `Matrix` class (`android.graphics.Matrix`) to support these graphic transformations.

You might have noticed the `Rotate` button in the earlier images. As a simple example of using a `Matrix`, the `Hour21ImageView` app implements a `Button` that rotates the image in the `ImageView` by 30 degrees.

Listing 21.2 shows the `onClickListener()` method for the `rotate` `Button`. On line 3, the `Matrix` associated with the `ImageView` is obtained. On line 4, you use the `setScaleType()` method to set the `ScaleType` to `MATRIX`. You must set `ImageView.ScaleType.MATRIX` in order to use a modified matrix on the `ImageView`. Line 5 is a matrix instruction to rotate the image 30 degrees around the point that is the center of the `ImageView`. Line 6 shows the matrix set for the `ImageView`. You must set `ImageView` `ScaleType` to `MATRIX` for this to take effect.

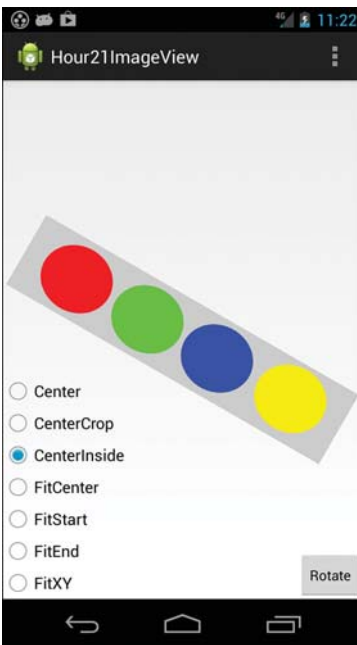
Figure 21.4 shows the rotated image.

**LISTING 21.2**    Rotating an Image

---

```
1: rotate.setOnClickListener(new OnClickListener() {
2:     public void onClick(View v) {
3:         Matrix matrix = imageView.getImageMatrix();
4:         imageView.setScaleType(ImageView.ScaleType.MATRIX);
5:         matrix.postRotate(30, imageView.getWidth()/2, imageView.getHeight()/2);
6:         imageView.setImageMatrix(matrix);
7:     }
8: });
```

---



**FIGURE 21.4**  
Rotated image

You can create complex transformations using `Matrix`. This app is a simple introduction to using a `Matrix` and the `MATRIX` `ScaleType`.

**NOTE**

---

**Rotation in Honeycomb**

Since API Level 11 (Honeycomb), the `View` class includes support for rotation. The methods `setPivotX()`, `setPivotY()`, and `setRotation()` are supported. The `Matrix` class supports more complex transformations.

---

## Setting Alpha

Alpha level indicates the opacity of an image. An image can be completely transparent, completely opaque, or somewhere in the middle. You can set the alpha level on an `ImageView` using the `setAlpha()` method or, since API level 11, by using the `setImageAlpha()` method. These methods take an integer parameter. A parameter of 0 indicates complete transparency and 255 indicates complete opacity.

## Bitmaps and Canvas

The `Bitmap` (`android.graphics.Bitmap`) class represents a bitmap image. You create bitmaps via the `BitmapFactory` (`android.graphics.BitmapFactory`) class.

Using a `BitmapFactory`, you can create bitmaps in three common ways: from a resource, a file, or an `InputStream`. To create a bitmap from a resource, you use the `BitmapFactory` method `decodeResource()`:

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.someImage);
```

To create a bitmap from a file or `InputStream`, you use the `decodeFile()` and `decodeStream()` methods, respectively.

## Handling Large Images

There are techniques for avoiding the dreaded OOM (out of memory) exception. Large images can have a significant impact on memory use in your app. To demonstrate this, in this section, you create an unrealistically large image to display in an `ImageView`. When the image is just loaded into the `ImageView`, the app fails with an out-of-memory error. A `java.lang.OutOfMemory` exception occurs. You'll learn to fix the memory error for this case.

You want to display the image at an appropriate size and resolution for the device. No point exists in showing a 10-foot mural in a 6-inch frame. Similarly, no point exists in showing a 20-inch image on a 3-inch phone screen. You can manipulate the image to display well and save memory.

The details of your app will influence your memory usage and the techniques that will work best in your case. This example shows how to handle a single large image.

As a demonstration, start with an image and increase it to an unrealistic size. This example uses a photo that is 72 inches x 54 inches and that has a 28Mb file size.

The image is in the `drawable` resource folder and has the id `R.drawable.largeimage`.

You can cause the app to fail with an out-of-memory error by trying to set an `ImageView` to this resource. You have an `ImageView` named `largeImage`. This line of code causes the app to fail:

```
largeImage.setImageResource(R.drawable.largeimage);
```

Some work is required to fix this, but handling an image this large is possible. In all cases, working with appropriately sized images would be ideal, but that does not always happen.

The goal is to get the dimensions of the underlying bitmap without actually rendering it. Getting those dimensions is not a memory-intensive activity. After you have the bitmap, you can determine an appropriate size for the bitmap that will fit in your display. If you have a 20-inch image and a 4-inch display, you request that the bitmap that is created in memory be created at a size and resolution that is appropriate for the 4-inch display.

## Using `BitmapFactory.Options`

You use the `BitmapFactory.Options` class with the `BitmapFactory` class; it is essential to how you handle large Bitmaps.

You use the following options from the `BitmapFactoryOptions` class:

- ▶ `inJustDecodeBounds`: If set to `true`, this option indicates that the bitmap dimensions should be determined by the `BitmapFactory`, but that the bitmap itself should not be created. This is the key to getting the bitmap dimensions without the memory overhead of creating the bitmap.
- ▶ `outWidth`: The width of the image set when you use `inJustDecodeBounds`.
- ▶ `outHeight`: The height of the image set when you use `inJustDecodeBounds`.
- ▶ `inSampleSize`: This integer indicates how much the dimensions of the bitmap should be reduced. Given an image of 1000x400, an `inSampleSize` of 4 will result in a bitmap of 250x100. The dimensions are reduced by a factor of 4.

Listing 21.3 shows the code to address handling large images. We'll step through the approach and the code. The code is in the `Hour21LargeImage` project in `MainActivity.java`.

---

### LISTING 21.3    **Displaying a Large Image**

```

1:  ImageView largeImage = (ImageView) findViewById(R.id.imageView1);
2:  Display display = getWindowManager().getDefaultDisplay();
3:  int displayWidth = display.getWidth();
4:  BitmapFactory.Options options = new BitmapFactory.Options();
5:  options.inJustDecodeBounds = true;
6:  BitmapFactory.decodeResource(getResources(), R.drawable.largeimage, options);
7:  int width = options.outWidth;
8:  if (width > displayWidth) {
9:      int widthRatio = Math.round((float) width / (float) displayWidth);
10:     options.inSampleSize = widthRatio;
11:  }
```

```

12: options.inJustDecodeBounds = false;
13: Bitmap scaledBitmap = BitmapFactory.decodeResource(getResources(),
14: R.drawable.largeimage, options);
15: largeImage.setImageBitmap(scaledBitmap);

```

---

Lines 2 and 3 get the size of the device display. You use this as the target size for reducing the image size.

In lines 4–7, you determine the size of the current bitmap. You do that by creating a `BitmapFactory.Options` class and setting the `inJustDecodeBounds` value to `true`. On line 6, the bitmap is decoded to get the dimensions. This is where you get the dimensions without the memory overhead of creating the bitmap. The result is available in `options.outWidth`. On line 7, you assign `options.outWidth` to the `int` variable `width`.

This example uses a simple test for the size of the image. Line 8 checks whether the width of the bitmap is greater than the size of the display. If that is the case, you determine the `inSampleSize` to use. You do that on lines 9 and 10. If the width of the bitmap is 1000 pixels and the size of the display is 250 pixels, you get an `inSampleSize` of 4 by dividing the width of the bitmap by the width of the display. For simplicity, this example does not check the height, which could theoretically leave you exposed to a bitmap that was 20 inches tall and 2 inches wide.

With the `inSampleSize` set to an appropriate value, you can render the image.

On line 12, the `inJustDecodeBounds` value is set to `false`. You want to decode the image and create the `Bitmap` object.

Lines 13 and 14 use the `BitmapFactory.decodeResource()` method to create a `Bitmap` and assign it to the variable `scaledBitmap`. Note that in this call, the `BitmapFactory.Options` variable `options` is passed as a parameter. That is how you indicate to the `BitmapFactory` what `inSampleSize` to use.

Displaying a 72-inch image on a phone is certainly not recommended, but Figure 21.5 shows that it can be done.



**FIGURE 21.5**  
Very large photo displayed on device

## Drawing Directly on a Canvas

You can do one more thing with an `ImageView` and bitmap. You can create a bitmap and draw directly on the `Canvas` (`android.graphics.Canvas`) that is associated with the bitmap. A `Canvas` is an object that you can draw on by calling drawing commands.

To see this, add another `Button` to the `MainActivity` activity from the `Hour21ImageView` project. Listing 21.4 shows the `onClickListener()` method for the draw button. The code creates a bitmap, gets the canvas, and draws the word “Hello” on the canvas. The resulting bitmap appears in the `ImageView`.

You create a new `Bitmap` on lines 3 and 4 by using the method `Bitmap.createBitmap()`. You set the width and height of the bitmap to that of the `ImageView` and use the `Bitmap.Config` (`android.graphics.Bitmap.Config`) set to `Bitmap.Config.ARGB_8888`.

The documentation for the `Bitmap` class offers a number of `createBitmap()` methods that take different parameters. These methods may return a mutable or an immutable `Bitmap`, but only a mutable `Bitmap` can be used for drawing.

On line 5, a Canvas is instantiated based on the bitmap that you created.

You apply simple drawing commands to the Canvas on lines 6–10. You create a Paint object, set the Color to blue, and set the text size. Line 6 gets the density of the display, which is used to get the text size you want. Hour 4 covered converting density independent pixels to pixels. Line 10 draws the word “Hello” in the center of the ImageView.

Line 11 updates the ImageView to show the generated bitmap.

---

#### LISTING 21.4 Drawing on a Canvas

```

1: draw.setOnClickListener(new OnClickListener() {
2:     public void onClick(View v) {
3:         Bitmap imageBitmap = Bitmap.createBitmap(imageView.getWidth(),
4:           imageView.getHeight(), Bitmap.Config.ARGB_8888);
5:         Canvas canvas = new Canvas(imageBitmap);
6:         float scale = getResources().getDisplayMetrics().density;
7:         Paint p = new Paint();
8:         p.setColor(Color.BLUE);
9:         p.setTextSize(24*scale);
10:        canvas.drawText("Hello", imageView.getWidth()/2, imageView.getHeight()/2,
11:          p);
12:        imageView.setImageBitmap(imageBitmap);
13:    }
14: });

```

---

## Using VideoViews

You use VideoViews (`android.widget.VideoView`) to play videos. In this section, you create a simple app in which a video is served from a web URL and shown in a VideoView. You have the option to control the video with controls that you create or with a MediaController (`android.widget.MediaController`). The example app uses both methods. First we’ll look at the basics.

A VideoView layout is often simple. Listing 21.5 shows a typical example. The desire to show a full screen video is common. The Hour21VideoView project contains this source code.

---

#### LISTING 21.5 VideoView Layout

```

1: <VideoView android:id="@+id/VideoView01"
2:   android:layout_height="match_parent"
3:   android:layout_width="match_parent">
4: </VideoView>

```

---



## Loading a Video

After you declare the `VideoView`, you need to give the view a video to play and start the video. A video may be read from a local file or from a remote server. In both cases, you can use the `setVideoUri()` method of `VideoView`. That method takes a URI as a parameter. Creating a URI from a string is typical. Listing 21.6 shows a `Uri` class being defined from a `String` and the resulting `Uri` being set to the `VideoView` in line 3.

---

### LISTING 21.6    Assigning a Video to a VideoView

---

```
1: String videoToPlay = "http://bffmedia.com/bigbunny.mp4";
2: Uri videoUri = Uri.parse(videoToPlay);
3: videoView.setVideoURI(videoUri);
```

---

You could use the same code in Listing 21.6 for reading a file from the SD card. Line 1 would change to include the location of a file:

```
String videoToPlay= Environment.getExternalStorageDirectory()+ "/Android/data/com.
bffmedia/videos/bigbunny.mp4";
```

`Environment.getExternalStorageDirectory()` refers to the location of the SD card.

## Starting, Pausing, and Positioning a Video

For controlling a video, the `VideoView` includes methods called `start()`, `pause()`, and `seekTo()`. The `start()` and `pause()` methods start and stop the video. The `seekTo()` method positions the video at a specific location and is based on milliseconds. Calling `seekTo()` with 10,000 positions the video at the tenth second. You can get the current position and duration of the video with the methods `getCurrentPosition()` and `getDuration()`. To skip ahead 10 seconds in the video, you use the following:

```
mVideoView.seekTo(mVideo.getCurrentPosition + 10000);
```

## Listening for the States of a VideoView

Two listeners are unique to a `VideoView`: `onPreparedListener()` and `OnCompletionListener()`. Videos do not start playing immediately. First, they are downloaded and buffered. That is where the `onPreparedListener()` comes in. You can do something in your user interface before the video begins. When it is ready, you can do something else. For example, you can show a progress bar before the video starts and then hide it in the `onPreparedListener()`.

The `onCompletionListener()` triggers when the video is done playing. It is an opportunity to repeat the video, start a new video, or change the user interface to prompt the user about what to do next.

To demonstrate the use of `VideoViews`, in this section you create an app with two activities. In `MainActivity.java`, there is a `VideoView`, a pause/play button, and a full screen button. The play/pause button controls the video. The full screen button opens the second activity. That activity, `VideoActivity.java`, plays the video in full screen and attaches a `MediaController` that provides native controls. Both `MainActivity.java` and `VideoActivity.java` are in the `Hour21VideoView` project.

You'll use the `OnPreparedListener()` to know when the video is ready to play. Until that time, you'll show a progress bar.

Listing 21.7 shows how to show a video. The `OnPreparedListener()` code begins on line 6. If the video is ready, the progress bar is hidden and the video plays. This occurs on lines 8–10.

---

### LISTING 21.7 Showing a Video

---

```

1: final VideoView videoView = (VideoView) findViewById(R.id.videoView);
2: final ProgressBar progressBar = (ProgressBar) findViewById(R.id.progressBar);
3: String videoToPlay = "http://bffmedia.com/bigbunny.mp4";
4: Uri video = Uri.parse(videoToPlay);
5: videoView.setVideoURI(video);
6: videoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener() {
7:     public void onPrepared(MediaPlayer mp) {
8:         progressBar.setVisibility(View.GONE);
9:         videoView.requestFocus();
10: videoView.start();
11:     }
12: });

```

---

In `MainActivity.java`, you can implement a play/pause button. To control the video, you use the `start()` and `pause()` methods of the `VideoView`.

The full screen button opens the `VideoActivity`. The code for `VideoActivity` is similar to the `MainActivity` code, but a `MediaController` is implemented.

Listing 21.8 shows the `onCreate()` method of `VideoActivity()`. A `MediaController` is defined in line 5. On line 6, that `MediaController` is set as the anchor of the `VideoView`, and on line 9, the `VideoView` sets its `MediaController` to the defined `MediaController`. This code creates and shows the `VideoView` with a `MediaController` attached.

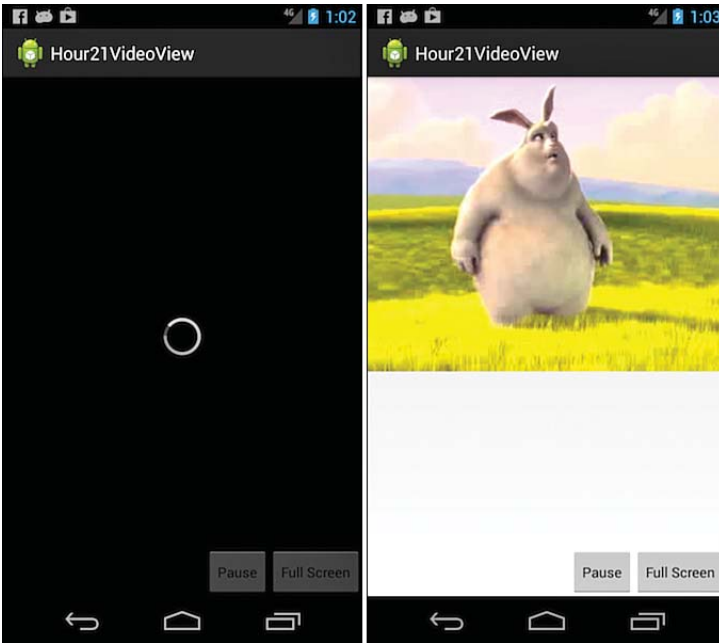
See Figures 21.6 and 21.7 to see how these two activities show videos.

**LISTING 21.8** Showing a Video with MediaController

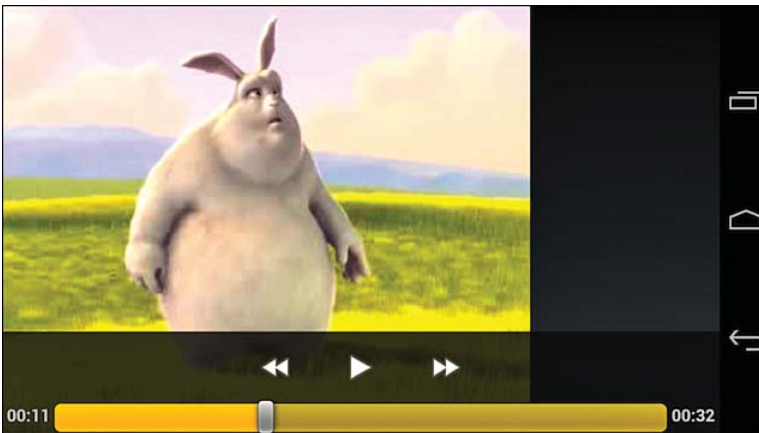
```

1: protected void onCreate(Bundle savedInstanceState) {
2:     super.onCreate(savedInstanceState);
3:     setContentView(R.layout.activity_video);
4:     VideoView videoView = (VideoView) findViewById(R.id.videoView);
5:     MediaController mediaController = new MediaController(this);
6:     mediaController.setAnchorView(videoView);
7:     String videoToPlay = "http://bffmedia.com/bigbunny.mp4";
8:     Uri video = Uri.parse(videoToPlay);
9:     videoView.setMediaController(mediaController);
10:    videoView.setVideoURI(video);
11:    videoView.start();
12: }

```



**FIGURE 21.6**  
VideoView with a custom pause button

**FIGURE 21.7**

VideoView with MediaController

## Playing Audio with MediaPlayer

Now consider a simple example of playing an .MP3 audio file using a `MediaPlayer` (`android.media.MediaPlayer`). When you do this, you are responsible for the state of the `MediaPlayer`. A `MediaPlayer` can be `reset()` and must be released when you are done using it.

The code for this example is in the `Hour21Audio` project in `MainActivity.java`. In the example, you read a file called `helloworld.mp3` from the `assets` directory. You do this in the `onResume()` method. The `MediaPlayer` is released in the `onPause()` method.

Listing 21.9 shows the `onResume()` method of `MainActivity.java`. The audio file is read from the `assets` folder and played using a `MediaPlayer`.

### LISTING 21.9 Playing an MP3 Audio File

```
1: protected void onResume() {
2:     super.onResume();
3:     try {
4:         audioFileDescriptor = getAssets().openFd("helloworld.mp3");
5:         mediaPlayer.setDataSource(audioFileDescriptor.getFileDescriptor(),
6:             audioFileDescriptor.getStartOffset(), audioFileDescriptor.getLength());
7:         mediaPlayer.prepare();
8:         mediaPlayer.start();
9:     } catch (IOException e) {
10:        e.printStackTrace();
11:    }
12: }
```

## Exploring More Media Options

This hour considered `ImageViews`, `Bitmaps`, `VideoViews`, and `MediaPlayer` and covered these components at a basic level. If you are creating complex media apps, you will want to dig deeper.

In addition, you might want to explore other media-related topics, including the following classes in Android:

- ▶ `AudioManager`: `android.media.AudioManager`
- ▶ `AudioFocus`: `android.media.AudioManager.OnAudioFocusChangeListener`
- ▶ `SoundPool`: `android.media.SoundPool`
- ▶ `AudioTrack`: `android.media.AudioTrack`
- ▶ `MediaPlayer`: `android.media.MediaPlayer`
- ▶ `Presentation`: `android.app.Presentation`

## Summary

This hour took a close look at `ImageViews` and bitmaps, including how to handle large images, and offered a basic approach to drawing directly on a canvas. The hour covered how to use `ScaleType` in detail and introduced the `Matrix` class to show a simple rotation of an image in an `ImageView`. This hour also introduced video and audio concepts using `VideoView` and `MediaPlayer`. Android provides significant ability to record and play various media types. This hour showed the basics and introduced the next steps to take to make great media-based apps.

## Q&A

- Q.** If I am developing an app that places images in a `ListView` or `GridView`, should I use `BitmapFactory.Options` to check the image size for each image?
- A.** If you do not have control of the size of the images coming from the server, then checking size is important. If you do have control over the images, then the ideal scenario is to have appropriately sized images. You can also check out open source projects such as `Thumbor` for resizing images from a server and `Picasso` for handling images in code. See <https://github.com/globocom/thumbor/wiki> and <https://github.com/square/picasso>.
- Q.** Is `VideoView` the only way to play a video on Android?
- A.** No, `VideoView` is one of the easier ways to play a video. You can also play a video via an `ActionView` intent or by using `MediaPlayer`. To show a video using an intent, create a `Uri` and an `Intent` as follows:

```
Intent intent = new Intent();
intent.setAction(Intent.ACTION_VIEW);
intent.setDataAndType(videoUri, "video/mp4");
startActivity(intent);
```

## Workshop

### Quiz

1. What is the purpose of `inJustDecodeBounds`?
2. How is `onPreparedListener()` used?
3. What does *mutable* mean?

### Answers

1. In `BitmapFactory.Options`, you use `inJustDecodeBounds` to decode a `Bitmap` to get the dimension, but not actually create a `Bitmap` in memory.
2. In a `VideoView`, `onPreparedListener()` listens for when a video is ready to play. The example in this hour showed a progress bar until the video was prepared and then displayed the video.
3. *Mutable* means *changeable*, which is important when you create `Bitmaps`. Some methods return mutable `Bitmaps` and others return immutable `Bitmaps`.

## Exercise

Add a `Button` in the `Hour21ImageView` project to set the alpha level. It will be like the rotation example. When the user clicks on the button, the alpha level should change.

You can use the other examples in this hour as starting points for what you would like to do in media.

*This page intentionally left blank*

*This page intentionally left blank*



# Index

## A

- accelerometer, 393**
- accessing**
  - Android Virtual Device Manager, 5
  - images, 53
- action bar, 68-70, 120**
  - ActionBar tabs for Photos and Favorites, 270-272
  - drop-down navigation, 120-121
  - fragments, 134-136
  - tab navigation, 122-124
  - when to use, 124-125
- ACTION\_EDIT, 296**
- ACTION\_INSERT, 296**
- ACTION\_INSERT\_OR\_EDIT, 296**
- ACTION\_PICK, 296**
- ACTION\_SEND, 35-37**
- ActionBar. See action bar**
- ActionBarSherlock, 125, 391**
- actions. See also action bar**
  - animations in, 388-390
  - coding, 15-17
  - in loaders, 247
- activities. See also intents**
  - back stack, 26-27
  - CustomCameraActivity.java, 341-345
  - CustomVideoActivity.java, 345
  - explained, 19-20
  - fragment and activity interaction, 136-138
  - Launch, 27-28
  - lifecycle, 19, 38-40
  - passing information between, 28
    - extras and bundles, 28-30
    - intents, 37
    - returning results, 30-33
  - preference activities, generating, 183
  - SettingsActivity, creating, 183
  - starting
    - back stack, 26-27
    - Launch activity, 27-28
    - starting one activity from another, 20-25
    - starting IntentService from, 386
- activity\_main.xml file, 45**
- adapters, 104-105**
  - custom adapters, 253
    - BaseAdapter, 253-254
    - custom CursorAdapters, 257
    - ViewHolder pattern, 255-256
  - ListCursorAdapter
    - bindView() method, 264-266, 270
    - getFlinging() method, 266-267
    - image loading, 267-269
    - newView() method, 263
    - onClick() event handling, 269-270
  - sync adapters, 285
- AdapterViewFlipper, 166-167**
- Add() method, 130**
- AddToBackStack() method, 131**
- ad-supported apps, 409**
- ADT (Android Developer Tools) bundle**
  - downloading, 3-4
  - folder structure, 4
- alert dialogs, 151-154**
  - creating, 151-152
  - custom views, 154
  - lists, 152-153
- AlertDialog, 151-154**
  - creating, 151-152
  - custom views, 154
  - lists, 152-153
- alerts, ProximityAlerts, 311-312**
- alpha level, 353**
- alternative resources, 63-64**
- Amazon, publishing apps on, 408**
- Android 4.2 folder, 44**
- Android Beam, 396**
- Android Compatibility package, 70-71**
- Android dependencies folder, 44, 46**
- Android developer documentation, 393**
- Android Developer Tools bundle. See ADT (Android Developer Tools) bundle**
- Android Device Chooser, 73**
- Android history, 59-60**
- Android Studio, 3**

**Android versions**

- action bar, 68-70
- fragments, 68-70
- strategy for device support, 72
- support library, 70-71
- table of, 68

**Android Virtual Device. See AVD (Android Virtual Device)****Android Virtual Device Manager, accessing, 5****android:configChanges, 66-67****AndroidManifest.xml file, 43**

- content providers, 231
- Facebook apps, 372
- intent filters, 35
- preparing for release, 400-401
- service definitions, 385

**animations, adding, 55, 388-390****API Explorer, 190-191****APIs (Application Programming Interfaces), 189**

- API levels, 68
- calendar API, 283-284
  - Calendar Content Provider, 285-288
- calendar events, listing, 289-292
- calendar updates with intents, 293
- CalendarContract tables, 284
- event data, accessing, 292-293
- contacts API, 294-295
  - contact intents, 296-297
  - ContactsContract class, 294-295
  - querying contacts, 295-296
- Maps API, 311
- remote APIs, 189
  - background downloads with AsyncTask, 197-200
  - connectivity, checking, 202
  - JSON-formatted data, 194-197
  - lists, displaying in fragments, 200
  - remote data, fetching, 190-193
- in-app payments, 410

**Application Programming Interfaces (APIs). See APIs (Application Programming Interfaces)****application resources, referencing, 47-48****apps**

- ad-supported apps, 409
- APIs (Application Programming Interfaces). *See* APIs (Application Programming Interfaces)
- in-app payments, 410
- Bluetooth support, 396
- camera app, 338
  - associating camera with SurfaceView, 339-340
  - CameraPreview class, 338-340
  - CustomCameraActivity.java, 341-345
  - CustomVideoActivity.java, 345
  - displaying previews, 341-343
  - permissions for capturing media, 338
  - SurfaceHolder.Callback implementation, 340
- development
  - determining app functionality, 259-263
  - planning, 262-263
  - wireframe diagrams, 260-262
- Facebook photo upload app
  - app IDs, 372
  - app implementation, 374-375
  - creating, 366-368
  - Facebook permissions, 373
  - Facebook requests, 375-376
  - login and session management, 374
- Flickr app. *See* Flickr app
- free apps, 409
- internationalization strategies
  - forgoing app internationalization, 324-325
  - implementing robust internationalization, 325-326
  - limiting app internationalization, 325

**launching**

- on devices, 72-74
  - with intents, 38
- monetizing, 409-410
- NFC (Near Field Communication), 396
- open source
  - ActionBarSherlock, 391
  - including in apps, 392
  - NineOldAndroids, 390-391
  - Picasso, 391-392
  - ViewPagerIndicator, 391
- OpenGL ES graphics API, 395
- paid apps, 409
- preparing for release, 399
  - Android manifest file, 400-401
  - signing apps, 401-405
- projects
  - creating, 6-8, 21
  - files at project creation, 43-45
  - folders at project creation, 43-45
- publishing, 405
  - on Amazon, 408
  - on Google Play, 405-408
  - OUYA and other markets, 409
- responsive apps with IntentService, 383-388
- reviewing in multiple languages, 322
- running, 11-12
- sensors, 393
- signing, 401-405
  - certificate file export, 404-405
  - with command-line tools, 403-404
  - with Eclipse, 402
  - package testing, 405
- source code, viewing, 10-11
- speech to text conversion, 395
- styles, 394-395
- text to speech (TTS) conversion, 395
- themes, 394-395
- user interface. *See* user interface
- views. *See* views
- XML layout, 10-11

ArrayAdapter, 104-105  
 arrays, JSONArrays, 195  
 assets folder, 44, 56-57  
 associating camera with  
 SurfaceView, 339-340  
 AsyncTask class, 108-109,  
 197-200

AsyncTaskLoader class, 246  
 attendees (event), returning list  
 of, 292  
 audio, playing with MediaPlayer,  
 361  
 AudioFocus class, 362  
 AudioManager class, 362  
 AudioTrack class, 362  
 AutoCompleteTextView, 106  
 AVD (Android Virtual Device), cre-  
 ating, 5, 11-12  
 avoiding OOM (out of memory)  
 exception, 353-355

## B

background downloads with  
 AsyncTask, 197-200  
 BaseAdapter, 253-254  
 Beam, 396  
 bin folder, 45  
 bindView() method, 257, 264-266,  
 270  
 Bitmap class, 353-355  
 BitmapFactory class, 353  
 BitmapFactoryOptions class,  
 354-355  
 Bluetooth support, 396  
 BroadcastReceiver, 386-388  
 build dependencies, resolving,  
 370  
 bundles, adding to intent, 28-30  
 Button control, 100-103  
 buttons, 15-17, 100-103

## C

cached images, saving, 222-224  
 calendar API, 283-284  
   Calendar Content Provider,  
   285-288

calendar events, listing,  
 289-292  
 calendar updates with intents,  
 293  
 CalendarContract tables, 284  
 event data, accessing,  
 292-293

**Calendar Content Provider,**  
**285-288**

calendar events, listing, 289-292

CalendarContract tables, 284

CalendarListFragment, 286

camera app, 338

  associating camera with  
   SurfaceView, 339-340

  CameraPreview class,  
   338-340

  CustomCameraActivity.java,  
   341-345

  CustomVideoActivity.java, 345  
   displaying previews, 341-343

  permissions for capturing  
   media, 338

  SurfaceHolder.Callback imple-  
   mentation, 340

  taking pictures, 343-345

**CameraPreview class, 338-340,**  
**341-343**

cameras, 329. *See also* media  
 camera app, 338

  associating camera with  
   SurfaceView, 339-340

  CameraPreview class,  
   338-340

  CustomCameraActivity.  
   java, 341-345

  CustomVideoActivity.java,  
   345

  displaying previews,  
   341-343

  permissions for capturing  
   media, 338

  SurfaceHolder.Callback  
   implementation, 340

  taking pictures, 343-345

intents

  advantages of, 338

  taking photos with,  
   332-336

  taking video with, 336-338

launching with intents,  
 333-334

  media capture, 329-330  
   media scanner, invoking,  
   331-332  
   media storage file, specify-  
   ing, 330-331  
   URLs (Uniform Resource  
   Identifiers), creating,  
   331-332

**Canvas, drawing on, 356-357**

**capturing media, 329-330**

  with intents, 336-338

  media scanner, invoking,  
   331-332

  media storage file, specifying,  
   330-331

  photos. *See* photos

  URLs (Uniform Resource  
   Identifiers), creating,  
   331-332

**certificate file, exporting, 404-405**

**changing**

  LinearLayout control proper-  
   ties, 85-86

  locales, 319

**CheckBoxPreference, 175-177**

**checking connectivity, 202**

**child views**

  layouts, 86-88

    gravity, 88

    layout margins, 87

    padding, 86-87

  weight, 88

**classes. *See* individual classes**

**close() method, 208-209**

**closing dialogs, 144**

**cloud access. *See* remote APIs**

**colors, 50**

**command-line tools**

  jarsigner, 403-404

  keytool, 403-404

**Commit() method, 131**

**compress() method, 224**

**connectivity, checking, 202**

**constants, creating, 30**

**contact intents, 296-297**

**ContactListFragment, 296**

**contacts API, 294-295**

  contact intents, 296-297

  ContactsContract class,  
   294-295

  querying contacts, 295-296

**Contacts table, 295**

**ContactsContract class, 294-295**

**contains() method, 173**

**content providers, 227**

adding to apps, 236-237

building, 228-235

AndroidManifest.xml file,  
231

declarations, 229-230

delete() method, 235

getType() method, 233-234

insert() method, 234-235

query() method, 231-233

shell, 228-229

update() method, 235

requesting files from, 237-242

file support implementa-  
tion, 238-239

image retrieval, 239-241

Uniform Resource Identifiers  
(URIs), 227-228

**ContentObserver, 241-242**

**controls**

adapters, 104-105

input controls, 98

buttons, 100-103

EditText, 98-100

TextView, 98-100

layout controls, 82-83

FrameLayout, 89

LinearLayout, 84-86

RelativeLayout control,  
89-91

spinner control, 105-106

getting data from, 106

setting up, 105

**converting**

speech to text, 395

text to speech, 395

**createBitmap() methods, 356**

**createLoader() method, 247**

**createPhoto() method, 209-210**

**criteria for location providers,  
302-303**

**currencies, handling, 327**

**Currency class, 327**

**CursorAdapter**

with CursorLoader class,  
248-252

ListCursorAdapter

bindView() method,  
264-266, 270

getFlinging() method,  
266-267

image loading, 267-269

newView() method, 263

**CursorLoader class**

CursorAdapter, 248-252

overview, 246

**cursors, 212-215**

**custom adapters, 253**

BaseAdapter, 253-254

custom CursorAdapters, 257

ListCursorAdapter

bindView() method,  
264-266, 270

getFlinging() method,  
266-267

image loading, 267-269

newView() method, 263

onClick() event handling,  
269-270

ViewHolder pattern, 255-256

**custom buttons, 103**

**custom view controls, 395**

**custom views**

in alert dialogs, 154

ListFragments, 159-161

**CustomCameraActivity.java,  
341-345**

**CustomVideoActivity.java, 345**

## D

**data access**

content providers, 227

adding to apps, 236-237

building, 228-230, 235

requesting files from,  
237-242

Uniform Resource  
Identifiers (URIs),  
227-228

custom adapters, 253

BaseAdapter, 253-254

custom CursorAdapters,  
257

ViewHolder pattern,  
255-256

loaders

actions, 247

creating, 247

CursorLoader with

CursorAdapter, 248-252

explained, 245

initializing, 247

loader classes, 246

loader states, 246-252

resetting, 248

remote APIs, 189

background downloads

with AsyncTask, 197-200

connectivity, checking, 202

JSON-formatted data,  
194-197

lists, displaying in frag-  
ments, 200

remote data, fetching,  
190-193

SQLite databases. See  
SQLite databases

**Data table, 295**

**data types in SharedPreferences,  
173-174**

**DatabaseHelper class, 207**

**databases (SQLite). See SQLite  
databases**

**date picker dialogs, 146-149**

**date strings, formatting, 326**

**DateFormat class, 326**

**DatePickerDialog, 146-149**

**declarations for content providers,  
229-230**

**decodeFile() method, 223, 353**

**decodeResource() method, 353**

**decodeStream() method, 224,  
353**

**deep linking, 378**

**default resources, specifying, 321**

**defining IntentService, 385-386**

**DelayReceiver, 388**

**delete() method, 235**

**deletePhoto() method, 211-212**

**deleting photos from database,  
211-212**

**density (screen)**

density-independent pixels, 61

explained, 60

handling, 61-62

**designing layouts, 80**

**detecting device features, 67**

**detecting flinging, 266-267**

**determining**

- app functionality, 259-263
- location, 299-300
  - criteria for location providers, 302-303
  - last known location, 303-305
  - Location object, 302
  - location settings, 305
  - LocationManager, 300-301
- system locale, 325-326

**developer documentation, 393****developing apps**

- determining app functionality, 259-263
- Facebook development, 366
- planning, 262-263
- wireframe diagrams, 260-262

**development of Android, 59-60****devices**

- display, 60
  - orientation, 63-67
  - screen density, 60-62
  - screen size, 61-63
- features, detecting, 67-68
- launching apps on, 72-74
- platform versions and compatibility levels
  - action bar, 68-70
  - fragments, 68-70
  - strategy for device support, 72
  - support library, 70-71
  - table of Android versions, 68

**dialogs**

- alert dialogs, 151-154
  - creating, 151-152
  - custom views, 154
  - lists, 152-153
- closing, 144
- date picker dialogs, 146-149
- displaying, 142-143
- explained, 141-142
- opening, 144
- ProgressDialog, 146
- time picker dialogs, 149-150

**dimensions, 50-51****dismiss() method, 143****display, 60**

- orientation
  - explained, 65
  - handling, 65
- screen density
  - explained, 60
  - handling, 61-62
- screen size
  - explained, 61
  - handling, 61-63

**displaying**

- camera previews, 341-343
- dialogs, 142-143
- fragments
  - dynamically, 129-131
  - with layouts, 128-129
- options menus, 114-119
- photos
  - Flickr app, 275-276
  - ImageViews, 348

**displays, external, 396****documentation (Android), 393****doInBackground() method, 198-199, 223****Dom parser, 202****dots-per-inch (DPI), 61****downloading Facebook SDK, 368****DPI (dots-per-inch), 61****drawable resources, 51-52**

- in buttons, 102
- images
  - accessing, 53
  - supported image formats, 52
- ninepatch, 52
- shapes, 53-54

**drawable-\* folders, 45, 52****drawing on Canvas, 356-357****drop-down navigation, action bar, 120-121****E****Eclipse**

- Eclipse folder, 4
- running projects from, 73-74
- signing apps, 402

**editing layouts, 81****EditText control, 98-100****EditTextPreference, 179****elements**

- <include/>, 82
- <support-screens>, 63
- <uses-feature>, 67

**event handling, user gestures, 394****events, calendar events, listing, 289-292****explicit intents, 34****exporting certificate file, 404-405****external display, Presentation class, 396****extras**

- accessing from intent, 29
- adding to intent, 28-29
- naming conventions, 30

**F****Facebook apps**

- app IDs, 372
- creating, 366-368
- Facebook photo upload app
  - app IDs, 372
  - app implementation, 374-375
  - creating, 366-368
- Facebook permissions, 373
- Facebook requests, 375-376
- login and session management, 374

**Facebook SDK, 365**

- deep linking, 378
- downloading, 368
- explained, 365-366
- Facebook development websites, 366
- Facebook photo upload app
  - app IDs, 372
  - app implementation, 374-375
  - creating, 366-368
- Facebook permissions, 373

- Facebook requests, 375-376
  - login and session management, 374
  - including in projects, 370-373
  - installing, 368-370
  - libraries, creating, 379
  - overview, 378
- fade-in animations, 55**
- fadein.xml file, 55**
- favorites, handling, 276-277**
- Favorites fragment, 270-272**
- features of devices, detecting, 67-68**
- fetchFavorites() method, 212**
- fetching remote data, 190**
  - API calls, 190-191
  - app structure, 193
  - URLConnection, 192-193
- files**
  - activity\_main.xml, 45
  - AndroidManifest.xml, 43
  - AndroidManifest.xml file
    - content providers, 231
    - Facebook apps, 372
    - preparing for release, 400-401
    - service definitions, 385
  - certificate file, exporting, 404-405
  - fadein.xml, 55
  - lc\_launcher-web.png, 45
  - image files, saving, 218
    - cached images, 222-224
    - retrieving item from Flickr, 221-222
  - MainActivity.java, 44
  - media storage file, specifying, 330-331
  - MP3 audio files, playing with MediaPlayer, 361
  - Proguard-project.txt, 45
  - project.properties, 45
  - requesting from content providers, 237-242
    - ContentObserver, 241-242
    - file support implementation, 238-239
    - image retrieval, 239-241
  - resource files. *See* resources
  - R.java, 44-46
  - strings.xml, 45
  - styles.xml, 45
- filters (intent), 35**
- findViewById() method, 81**
- FiveSecondService.java, 385**
- flags (intent), 26-27**
- Flickr app. *See also* SQLite databases**
  - API Explorer, 190-191
  - background downloads with AsyncTask, 197-200
  - content providers
    - adding, 236-237
    - building, 228-235
    - ContentObserver, 241-242
    - requesting files from, 237-242
    - Uniform Resource Identifiers (URIs), 227-228
  - custom adapters, 253
    - BaseAdapter, 253-254
    - ViewHolder pattern, 255-256
  - development
    - determining app functionality, 259-263
    - planning, 262-263
    - wireframe diagrams, 260-262
  - final app inventory, 277
  - FlickrPhoto class, 205-206
  - FlickrPhotoDbAdapter, 206-209
  - JSON-formatted data, 194-197
  - ListCursorAdapter
    - bindView() method, 264-266, 270
    - getFlinging() method, 266-267
    - image loading, 267-269
    - newView() method, 263
    - onClick() event handling, 269-270
  - lists, displaying in fragments, 200
  - loaders
    - actions, 247
    - creating, 247
    - CursorLoader with CursorAdapter, 248-252
    - explained, 245
    - initializing, 247
    - loader classes, 246
    - loader states, 246-252
    - resetting, 248
- Photo and Favorite photo fragments, 270-272
- PhotoGridFragment, 272-275
- PhotoListFragment, 272-274
- photos
  - deleting, 211-212
  - displaying in list or grid, 272-275
  - displaying single image, 275-276
  - favorites, 276-277
  - getting from database, 212-215
  - image files, saving, 218-224
  - inserting into database, 209-210, 215-216
  - retrieving from Flickr, 221-222
  - titles, displaying, 216-218
  - updating, 210-211
  - remote data, fetching, 190-193
    - retrieving items from, 221-222
- FlickrPhoto class, 196-197, 206**
- FlickrPhotoDbAdapter class, 206-209, 215**
- FlickrPhotoProvider, 229-230**
  - adding to app, 236-237
  - building, 228-235
    - AndroidManifest.xml file, 231
    - declarations, 229-230
    - delete() method, 235
    - getType() method, 233-234
    - insert() method, 234-235
    - query() method, 231-233
    - shell, 228-229
    - update() method, 235
  - requesting files from, 237-242
    - ContentObserver, 241-242
    - file support implementation, 238-239
    - image retrieval, 239-241
    - Uniform Resource Identifiers (URIs), 227-228
- flinging, detecting, 266-267**
- flipping views, 166-167**
- folders**
  - Android 4.2, 44
  - Android dependencies folder, 44, 46
  - assets, 44, 56-57
  - bin, 45

- drawable-\*, 45
- Eclipse, 4
- libs, 44
- res, 11, 45
- sdk, 4
- src, 44
- forgoing app internationalization, 324-325
- formatting date and time strings, 326
- FormControlActivity class, 100
- fragments, 68-70
  - with action bars, 134-136
  - AdapterViewFlipper, 166-167
  - CalendarListFragment, 286
  - ContactListFragment, 296
  - content providers, adding, 236-237
  - displaying
    - dynamically, 129-131
    - layouts, 128-129
  - explained, 127-128
  - fragment and activity interaction, 136-138
  - galleries, 164-166
  - grid views, 162-163
  - ListFragments, 157
    - creating, 158-159
    - customizing, 159-161
  - lists in, 200
  - navigating between, 132-134
  - Photo and Favorite photo fragments, 270-272
  - PhotoGridFragment, 251-252, 272-275
  - PhotoListFragment, 200, 216-218, 272-274
  - PreferencesFragment
    - CheckBoxPreference, 175-177
    - creating, 174-175
    - EditTextPreference, 179
    - ListPreference, 177-179
    - reading preferences, 182
    - titles, adding, 180-181
    - support package, 131
- FragmentTransaction object, 130-131
- FrameLayout control, 82, 89
- free apps, 409
- Froyo, 68

## G

- galleries, 164-166
- Gallery widget, 164-166
- generating preference activities, 183
- geo intent, 309-310
- Geocoder class, 306-309
- geocoding, 306-309
- gestures, 394
- getActivity() method, 134
- getAll() method, 173
- getArguments() method, 138
- getBoolean() method, 172
- getCacheDir() method, 220
- getColor() method, 50
- getColumnIndex() method, 214
- getConfiguration() method, 326
- getContentResolver() method, 277
- getCount() method, 253
- getDialog() method, 142
- getDrawable() method, 53
- getExternalFilesDir() method, 220
- getExternalStorageDirectory() method, 220
- getExternalStoragePublicDirectory() method, 220, 330-331
- getFilesDir() method, 220
- getFlinging() method, 266-267
- getFloat() method, 173
- getFromLocation() method, 308
- getFromLocationName() method, 307
- getInt() method, 173, 214
- getItem() method, 253
- getItemId() method, 253
- getLastKnownLocation() method, 303
- getLatitude() method, 307
- getLoaderManager() method, 247
- getLocality() method, 309
- getLong() method, 173
- getLongitude() method, 307
- getPackageManager() method, 67-68
- getPhotoFromCursor() method, 218
- getPhotos() method, 197
- getProvider() method, 302-303
- getRecentLocation() method, 304
- getResources() method, 47-48
- getSelectedItem() method, 106
- getSelectedItemPosition() method, 106
- getSharedPreferences() method, 171-172
- getString() method, 173, 214
- getStringSet() method, 173
- getSupportFragmentManager() method, 131
- getSystem() method, 48
- getText() method, 99, 102
- getTime() method, 304
- getType() method, 233-234
- getView() method, 253, 255-256
- getWritableDatabase() method, 208-209
- Gingerbread, 68
- global positioning system. *See* GPS (global positioning system)
- Google Play, publishing apps to, 405-408
- Google Play services, 312-315
- Google Street View support, 310
- Google TV, 60
- GPS (global positioning system), 299. *See also* location-based services
  - determining location
    - criteria for location providers, 302-303
    - last known location, 303-305
    - Location object, 302
    - location settings, 305
    - LocationManager, 300-301
  - geo intent, 309-310
  - geocoding, 306-309
  - Google Play services, 312-315
  - Maps API, 311
  - ProximityAlerts, 311-312
- gradients, 53-54
- graphics
  - OpenGL ES graphics API, 395
  - photos. *See* photos
- gravity, 88
- grid views, 162-163, 272-275
- Groups table, 295
- Gson, 196

**H**

hiding images from media scanner, 331

hint property (EditText control), 98

hints, showing in EditText view, 98

history of Android, 59-60

Honeycomb, 68

HorizontalScrollView, 168

HTC Dream phone, 59

URLConnection, 192-193

**I**

lc\_launcher-web.png file, 45

Ice Cream Sandwich, 68

icons, adding to menu items, 117

IDs (app), 372

image files, saving, 218

    cached images, 222-224

    retrieving item from Flickr, 221-222

ImageButton control, 100-103

images

    accessing, 53

    Bitmap class, 353-355

    Canvas, drawing on, 356-357

    image files, saving, 218

        cached images, 222-224

        retrieving item from Flickr, 221-222

    ImageViews, 110, 347

        alpha level, 353

        displaying images, 348

        rotating images, 351-352

        ScaleTypes in, 348-351

    photos. *See* photos

    supported image formats, 52

ImageViews, 110, 347

    alpha level, 353

    displaying images, 348

    rotating images, 351-352

    ScaleTypes in, 348-351

implicit intents, 34-37

<include/> element, 82

initializing loaders, 247

initLoader() method, 247

inJustDecodeBounds option (BitmapFactoryOptions), 354

input controls, 98

    buttons, 100-103

    EditText, 98-100

    TextView, 98-100

InputStreamToString() method, 56

inputType property (EditText control), 99

inSampleSize option (BitmapFactoryOptions), 354

insert() method, 234-235

installing Facebook SDK, 368-370

IntelliJ, 3

intents

    ACTION\_SEND intent, handling, 35-37

    advantages of, 338

    bundles, adding, 28-30

    contact intents, 296-297

    creating, 22

    explained, 34-37

    explicit intents, 34

    extras

        accessing, 29

        adding, 28-29

        naming conventions, 30

    geo intent, 309-310

    implicit intents, 34-37

    intent filters, 35

    intent flags, 26-27

    launching apps with, 38

    launching cameras with, 333-334

    passing information with, 37

    taking photos with, 332-336

    taking video with, 336-338

    updating calendar with, 293

IntentService

    BroadcastReceiver, 386-388

    defining, 385-386

    explained, 383-384

    starting from activity, 386

interfaces

    onDateEnteredListener, 149

    onNavigationListener, 120-121

internationalization, 323-324. *See also* localization

    principles, 317-318

    strategies

        forgoing app internationalization, 324-325

    implementing robust internationalization, 325-326

    limiting app internationalization, 325

isFavorite value, 276-277

isOnline() method, 202

**J**

jarsigner, 403-404

Java code, viewing, 10-11

Jelly Bean, 68

JSONArrays, 195

JSON-formatted data, 194

    JSONArrays, 195

    JSONObjects, 194

    JsonReader, 195

    parsing, 196-197

JSONObjects, 194

JsonReader, 195

**K**

keytool, 403-404

Kindle Fire, 60, 72

**L**

language-specific resources, specifying, 321

large images, handling, 353-355

last known location, determining, 303-305

Launch activity, 27-28

launching

    apps, 11-12

        on devices, 72-74

        with intents, 38

    cameras with intents, 333-334

Layout Resource Editor, 80

layout\_gravity property, 88

layout\_weight property, 88

layouts

    child views, 86-88

    gravity, 88

    layout margins, 87



- padding, 86-87
  - weight, 88
  - designing, 80
  - editing, 81
  - explained, 79-80
  - for fragments, 128-129
  - FrameLayout control, 82, 89
  - Layout Resource Editor, 80
  - LinearLayout control, 82-84
    - changing properties of, 85-86
    - common attributes, 84
  - RelativeLayout control, 82, 89-91
  - TableLayout control, 82
  - using programmatically, 81
  - VideoViews, 357
  - libraries**
    - ActionBarSherlock, 391
    - creating, 379
    - Facebook SDK. *See* Facebook SDK
    - Gson, 196
    - support library, 70-71, 131
    - ViewPagerIndicator, 391
  - libs folder, 44**
  - lifecycle**
    - of activities, 19, 38-40
    - of services, 383
  - light sensor, 393**
  - limiting app internationalization, 325**
  - LinearLayout control, 82-84**
    - changing properties of, 85-86
    - common attributes, 84
  - links, deep linking, 378**
  - ListCursorAdapter**
    - BindView() method, 264-266, 270
    - getFlinging() method, 266-267
    - image loading, 267-269
    - newView() method, 263
    - onClick() event handling, 269-270
  - ListFragments, 157**
    - creating, 158-159
    - customizing, 159-161
    - PhotoListFragment, 200, 216-218, 236-237
  - listing calendar events, 289-292**
  - ListPreference, 177-179**
  - lists**
    - in alert dialogs, 152-153
    - CalendarListFragment, 286
    - ContactListFragment, 296
    - displaying in fragments, 200
    - ListFragments, 157
      - creating, 158-159
      - customizing, 159-161
    - PhotoListFragment, 200, 216-218
  - Loader class, 246**
  - LoaderManager class, 246**
  - LoaderManagerCallbacks class, 246**
  - loaders**
    - actions, 247
    - creating, 247
    - CursorLoader with CursorAdapter, 248-252
    - explained, 245
    - initializing, 247
    - loader classes, 246
    - loader states, 246-252
    - resetting, 248
  - LoadImage() method, 222**
  - loading**
    - images, 267-269
    - video, 358
  - locales**
    - changing, 319
    - default resources, 321
    - determining system locale, 325-326
    - explained, 317
    - handling, 317-320
  - localization, 317**
    - Android support for, 318
    - currencies, handling, 327
    - date and time string formatting, 326
    - general internationalization principles, 317-318
    - internationalization strategies, 323-324
      - forgoing app internationalization, 324-325
    - implementing robust internationalization, 325-326
    - limiting app internationalization, 325
    - language-specific resources, specifying, 321
    - locales
      - changing, 319
      - default resources, 321
      - determining system locale, 326
      - explained, 317
      - handling, 317-320
    - region-specific resources, specifying, 322
    - reviewing apps in multiple languages, 322
  - Location object, 302**
  - location providers, criteria for, 302-303**
  - location settings, 305**
  - location-based services, 299**
    - determining location, 299-300
      - criteria for location providers, 302-303
    - last known location, 303-305
    - Location object, 302
    - location settings, 305
    - LocationManager, 300-301
    - geo intent, 309-310
    - geocoding, 306-309
    - Google Play services, 312-315
    - Maps API, 311
    - ProximityAlerts, 311-312
  - LocationManager, 300-301**
  - logcat, monitoring, 12**
  - login management (Facebook apps), 374**
- M**
- magnetic field sensor, 393**
  - MainActivity.java file, 44**
  - makePhotoList() method, 197**
  - Manifest file**
    - content providers, 231
    - Facebook apps, 372
    - intent filters, 35

- preparing for release, 400-401
- service definitions, 385
- Maps API, 311**
- margins, child views, 87**
- Matrix class, 351-352**
- media, 347. See also cameras**
  - Android classes, 362
  - audio, playing with MediaPlayer, 361
  - Bitmap class, 353-355
  - Canvas, drawing on, 356-357
  - capturing, 329-330
    - with intents, 336-338
    - media scanner, invoking, 331-332
    - media storage file, specifying, 330-331
    - URLs (Uniform Resource Identifiers), creating, 331-332
  - ImageViews, 347
    - alpha level, 353
    - displaying images, 348
    - rotating images, 351-352
    - ScaleTypes in, 348-351
  - MediaController, 359
  - photos. *See* photos
  - VideoViews
    - layout, 357
    - listeners, 358-359
    - loading video, 358
    - showing video, 358-359
    - starting, pausing, and positioning video, 358
- media scanner, invoking, 331-332**
- media storage file, specifying, 330-331**
- MediaController, 359**
- MediaPlayer, 361-362**
- MediaStore, 329**
- menus, options menus, 113**
  - adding icons to, 117
  - displaying, 114-119
  - responding to, 119-120
  - when to use, 124-125
- methods. See individual methods**
- Miracast, 396**
- monetizing apps, 409-410**
- monitoring logcat, 12**

- moveToFirst() method, 214**
- moveToLast() method, 214**
- moveToNext() method, 214**
- moveToPosition() method, 214**
- moveToPrevious() method, 214**
- MP3 audio files, playing with MediaPlayer, 361**

## N

### naming conventions

- extras, 30
- packages, 7

### navigation

- action bar
  - drop-down navigation, 120-121
  - tab navigation, 122-124
  - when to use, 124-125
- buttons, 15-17, 100-103
- fragments
  - action bars, 134-136
  - navigating between fragments, 132-134
- options menus, 113
  - displaying, 114-119
  - responding to, 119-120
  - when to use, 124-125
- paging controls, 167-168

- Near Field Communication (NFC), 396**

- newView() method, 257, 263**

- NFC (Near Field Communication), 396**

- NineOldAndroids, 390-391**

- ninepatch, 52

- notifyChange() method, 219**

## O

- ObjectAnimator, 388-390**

- ODK (OUYA Developer Kit), 409**
- onActivityCreated() method, 163, 236-237, 240-241, 290**

- onActivityResult() method, 332, 335-336, 337-338**

- onAnswerSelected() method, 138**

- onAnswerSelectedListener() method, 136-137**

- onAttach() method, 137-138**

- onClick() method, 17**

- onClickListener() method, 17, 336-337**

- onCompletionListener() method, 358-359**

- onConfigurationChanged() method, 66-67**

- onCreate() method, 10, 24-25, 39-40, 207, 293**

- onCreateDialog() method, 142, 150, 154**

- onCreateLoader() method, 246, 272, 275, 287, 291**

- onCreateOptionsMenu() method, 10, 114-119**

- onCreateView() method, 159-167**

- onDateEnteredListener interface, 149**

- onDateSet() method, 146-149**

- onDestroy() method, 40, 218**

- onDoubleTap, 394**

- onDoubleTapEvent, 394**

- onDown, 394**

- onFling, 394**

- onHandleIntent() method, 383**

- onListItemClick() method, 159, 218, 288**

- onLoaderReset() method, 246, 248, 287**

- onLoadFinished() method, 246, 247, 275, 287**

- onLocationChanged() method, 300-301, 308**

- onLongPress, 394**

- onNavigationListener interface, 120-121**

- onOptionsItemSelected() method, 119-120**

- onPause() method, 39-40, 361, 387**

- onPostExecute() method, 199-200**

- onPreparedListener() method, 358-359**

- onResume() method, 39-40, 361, 387**

- onScroll, 394**

- onSessionStateChanged() method, 374**

onShowPress, 394  
 onSingleTapConfirmed, 394  
 onSingleTapUp, 394  
 onStart() method, 39-41  
 onStop() method, 39-41  
 onUpgrade() method, 207  
**OOM (out of memory) exception, avoiding, 353-355**  
 open() method, 208-209  
**open source**  
     ActionBarSherlock, 391  
     in apps, 392  
     NineOldAndroids, 390-391  
     Picasso, 391-392  
     ViewPagerIndicator, 391  
 openFile() method, 238  
 openFileInput() method, 219  
**OpenGL ES graphics API, 395**  
 opening dialogs, 144  
 openRawResource() method, 56  
**options menus, 113**  
     adding icons to, 117  
     displaying, 114-119  
     responding to, 119-120  
     when to use, 124-125  
**orientation, handling, 65-67**  
     android:configChanges, 66-67  
     retaining data across configuration changes, 66  
**orientation sensor, 393**  
**out of memory (OOM) exception, avoiding, 353-355**  
**outHeight option (BitmapFactoryOptions), 354**  
**outWidth option (BitmapFactoryOptions), 354**  
**OUYA Developer Kit (ODK), 409**

## P

**PackageManager class, 67-68**  
**packages**  
     Android Compatibility, 70-71  
     names, 7  
     testing, 405  
**padding child views, 86-87**  
**paging controls, 167-168**  
**paid apps, 409**

**ParcelFileDescriptor, 239**  
 parsing JSON-formatted data, 196-197  
**passing information between activities, 28**  
     extras and bundles, 28-30  
     intents, 37  
     returning results, 30-33  
**pausing video, 358**  
**permissions (Facebook), 373**  
**permissions for capturing media, 338**  
**personalizing**  
     actions, 15-17  
     user interface, 13-14  
**PhoneLookups table, 295**  
**PhotoContentObserver, 241-242**  
**photoDbAdapter variable, 215**  
**PhotoGridFragment, 251-252, 272-275**  
**PhotoListFragment, 200, 216-218, 236-237, 272-274**  
**photos**  
     ActionBar tabs for Photos and Favorites, 270-272  
     alpha level, 353  
     deleting, 211-212  
     displaying  
         ImageViews, 348  
         in list or grid, 272-275  
         photo titles from database, 216-218  
         single image, 275-276  
     Facebook photo upload app  
         app implementation, 374-375  
         Facebook permissions, 373  
         Facebook requests, 375-376  
         login and session management, 374  
     favorites, 276-277  
     getting from database, 212-215  
     image files, saving, 218  
         cached images, 222-224  
         retrieving item from Flickr, 221-222  
     inserting into database, 209-210, 215-216

loading with  
     ListCursorAdapter, 267-269  
 retrieving from Flickr, 221-222  
 retrieving from  
     FlickrPhotoProvider, 239-241  
 rotating, 351-352  
 taking, with camera app, 343-345  
 taking with intents, 332-336  
 titles, displaying, 216-218  
 updating in database, 210-211  
**Photos fragment, 270-272**  
**Picasso, 391-392**  
**PictureCallback() method, 344**  
**PieAdapter, 253-254**  
**planning apps, 262-263**  
**platform tools, 5**  
**platform versions and compatibility levels**  
     action bar, 68-70  
     fragments, 68-70  
     strategy for device support, 72  
     support library, 70-71  
     table of Android versions, 68  
**playing audio with MediaPlayer, 361**  
**positioning video, 358**  
**postPhoto() method, 375-376**  
**preference activities, generating, 183**  
**preferences, 171**  
     SharedPreferences, 171  
         data types, 173-174  
         methods, 173-174  
         reading from, 172-173  
         setting preferences, 171-172  
     user preferences, 174  
         CheckBoxPreference, 175-177  
         EditTextPreference, 179  
         ListPreference, 177-179  
         preference activities, 183  
         PreferencesFragment, creating, 174-175  
         reading, 182  
         titles, adding, 180-181

**PreferencesFragment**

- CheckBoxPreference, 175-177
- creating, 174-175
- EditTextPreference, 179
- ListPreference, 177-179
- reading preferences, 182
- titles, adding, 180-181
- preparing for release, 399**
  - Android manifest file, 400-401
  - signing apps, 401-405
    - certificate file export, 404-405
    - with command-line tools, 403-404
    - with Eclipse, 402
    - package testing, 405
- Presentation class, 362, 396**
- previews, displaying (camera), 341-343**
- ProgressDialog, 146**
- progress bars, 107, 109-110**
- ProgressBars, 107, 109-110**
- Proguard-project.txt file, 45**
- project.properties file, 45**
- projects**
  - Android dependencies folder, 46
  - creating, 6-8, 21
  - files
    - files at project creation, 43-45
    - R.java, 44-46
  - folders at project creation, 43-45
  - resources, 47
    - advantages of, 47
    - alternative resources, 63-64
    - animations, 55
    - application resources, referencing, 47-48
    - colors, 50
    - dimensions, 50-51
    - images, 52-53
    - ninepatch, 52
    - raw resource files, 56
    - shapes, 53-54
    - storing in assets folder, 56-57
    - strings, 49
    - styles, 55-56
    - system resources, referencing, 48
    - running from Eclipse, 73-74
- proximity sensor, 393**
- ProximityAlerts, 311-312**
- publishing apps, 405**
  - on Amazon, 408
  - on Google Play, 405-408
  - OUYA and other markets, 409
- putExtra() method, 37, 293, 387**

**Q**

- query() method, 231-233**
- querying**
  - contacts, 295-296
  - database data, 212-215

**R**

- raw resource files, 56**
- RawContacts table, 295**
- reading**
  - SharedPreferences, 172-173
  - user preferences, 182
- RECORD\_AUDIO permission, 338**
- rectangles, 53-54**
- recurring event definitions, 291**
- referencing**
  - application resources, 47-48
  - system resources, 48
- region-specific resources, specifying, 322**
- registerOnSharedPreferenceChangeListener() method, 174**
- registerReceiver() method, 388**
- RelativeLayout control, 82-83, 89-91**
- release, preparing apps for, 399**
  - Android manifest file, 400-401
  - signing apps, 401-405
    - certificate file export, 404-405
    - with command-line tools, 403-404
    - with Eclipse, 402
    - package testing, 405
- remote APIs, 189**
  - background downloads with AsyncTask, 197-200
  - connectivity, checking, 202
  - JSON-formatted data, 194
    - JSONArrays, 195
    - JSONObjects, 194
    - JsonReader, 195
    - parsing, 196-197
  - lists, displaying in fragments, 200
  - remote data, fetching, 190
    - API calls, 190-191
    - app structure, 193
    - URLConnection, 192-193
- remote data, fetching, 190**
  - API calls, 190-191
  - app structure, 193
  - URLConnection, 192-193
- Remove() method, 130**
- removeUpdates() method, 305**
- Replace() method, 130**
- requesting files from content providers, 237-242**
  - ContentObserver, 241-242
  - file support implementation, 238-239
  - image retrieval, 239-241
- requestLocationUpdates() method, 305**
- requests (Facebook), 375-376**
- res folder, 11, 45**
- reset() method, 361**
- resetting loaders, 248**
- resolving build dependencies, 370**
- resources, 11, 47**
  - advantages of, 47
  - alternative resources, 63-64
  - animations, adding, 55
  - application resources, referencing, 47-48
  - in buttons, 102
  - colors, 50
  - default resources, specifying, 321
  - dimensions, 50-51

- images
    - accessing, 53
    - supported image formats, 52
  - language-specific resources, specifying, 321
  - ninepatch, 52
  - raw resource files, 56
  - region-specific resources, specifying, 322
  - shapes, 53-54
  - storing in assets folder, 56-57
  - strings, 49
  - styles, 55-56
  - system resources, referencing, 48
  - results, returning, 30-33**
  - RetrievalImage() method, 239-240**
  - retrieving item from Flickr, 221-222**
  - returning results, 30-33**
  - reverse-geocoding, 308-309**
  - reviewing apps in multiple languages, 322**
  - R.java file, 44-46**
  - robust internationalization, 325-326**
  - rotating images, 351-352**
  - Run Configurations screen, 74**
  - running. See launching**
- S**
- saving image files, 218**
    - cached images, 222-224
    - retrieving item from Flickr, 221-222
  - SAX parser, 202**
  - ScaleTypes, 348-351**
  - screen density**
    - explained, 60
    - handling, 61-62
  - screen size**
    - explained, 61
    - handling, 61-63
      - alternative resources, 63-64
      - <support-screens> element, 63
  - scrolling controls, 167-168**
  - sdk folder, 4**
  - SeekBar, 107, 110**
  - sendBroadcast() method, 331, 386-388**
  - sensors, 393**
  - services**
    - explained, 383
    - Google Play services, 312-315
    - IntentService
      - BroadcastReceiver, 386-388
      - defining, 385-386
      - explained, 383-384
      - starting from activity, 386
    - lifecycle, 383
    - location-based services, 299
      - determining location, 299-305
      - geocoding, 306-309
      - translation services, 322
  - session management (Facebook apps), 374**
  - setCompoundDrawablesWithIntrinsicBounds() method, 102**
  - setContentView() method, 46, 81**
  - setData() method, 331**
  - setImageAlpha() method, 353**
  - setImageBitmap() method, 348**
  - setImageDrawable() method, 348**
  - setImageResource() method, 348**
  - setImageUri() method, 348**
  - setItems() method, 153**
  - setMessage() method, 153**
  - setOutputFile() method, 345**
  - setScaleType() method, 351**
  - setText() method, 102**
  - setting preferences, SharedPreferences, 171-172**
  - SettingsActivity, creating, 183**
  - setTitle() method, 153**
  - SetTransition() method, 131**
  - ShapeDrawable class, 53-54**
  - shapes, 53-54**
  - SharedPreferences, 171**
    - data types, 173-174
    - methods, 173-174
    - reading from, 172-173
    - setting preferences, 171-172
  - shootVideoButton, 336-337**
  - show() method, 144**
  - showAsAction attribute (onCreateOptionsMenu() method), 115-116**
  - showFragmentA() method, 132-134**
  - showFragmentB() method, 132-134**
  - showGrid() method, 273**
  - showing video, 358-359**
  - showList() method, 200, 271, 273**
  - signing apps, 401-405**
    - certificate file export, 404-405
    - with command-line tools, 403-404
    - package testing, 405
  - simple resources**
    - colors, 50
    - dimensions, 50-51
    - strings, 49
  - SimpleDateFormat() method, 331**
  - size of screen**
    - explained, 61
    - handling, 61-63
  - SoundPool class, 362**
  - source code, viewing, 10-11**
  - specifying**
    - default resources, 321
    - language-specific resources, 321
    - media storage file, 330-331
    - region-specific resources, 322
  - speech to text conversion, 395**
  - spinner control, 105-106**
    - getting data from, 106
    - setting up, 105
  - SQLite databases, 205**
    - adding to apps, 215-218
    - cursors, 212-215
    - PhotoListFragment, 216-218
    - photos
      - deleting, 211-212
      - image files, saving, 218-224
      - inserting, 209-210, 215-216
      - querying, 212-215
      - updating, 210-211

SQLiteOpenHelper, 206-209  
tables, 205-206

**SQLiteOpenHelper**, 206-209

**src folder**, 44

**startActivityForResult() method**, 30-33, 332, 375

**starting**

- activities
  - back stack, 26-27
  - Launch activity, 27-28
  - starting one activity from another, 20-25
- IntentService, 386
- video, 358

**startService() method**, 386

**states, loader states**, 246-252

**StatusUpdates table**, 295

**strings**

- data and time string formatting, 326
- explained, 49

**strings.xml file**, 45

**styles**, 55-56, 394-395

**styles.xml file**, 45

**support library**, 70-71, 131

**<support-screens> element**, 63

**surfaceChanged() method**, 340

**surfaceCreated() method**, 340

**surfaceDestroyed() method**, 340

**SurfaceHolder.Callback implementation**, 340

**SurfaceView**, associating camera with, 339-340

**swapCursor() method**, 247, 287

**sync adapters**, 285

**system locale**, determining, 325-326

**system resources**, referencing, 48

**T**

**tab navigation, action bar**, 122-124

**TableLayout control**, 82

**tables**, 205-206

- CalendarContract tables, 284
- ContactsContract tables, 295

**TabListener() method**, 122-124

**takePhotoButton**, 333-334

**takePicture() method**, 343-345

**taking photos**

- with camera app, 343-345
- with intents, 332-336

**taking video**. *See capturing media*

**temperature sensor**, 393

**testing packages**, 405

**text to speech (TTS) conversion**, 395

**TextView control**, 98-100

**themes**, 394-395

**time picker dialogs**, 149-150

**time strings, formatting**, 326

**TimePickerDialog**, 149-150

**titles, adding to user preferences**, 180-181

**translation services**, 322

**T-Store**, 409

## U

**UILifecycleHelper**, 374

**Uniform Resource Identifiers (URIs)**, 227-228, 331-332

**unregisterOnSharedPreferenceChangeListener() method**, 174

**unregisterReceiver() method**, 388

**update() method**, 235

**updatePhoto() method**, 210-211

**updating**

- calendar with intents, 293
- photos, 210-211
- user interface, 13-14

**uploads, Facebook photo upload app**

- app implementation, 374-375
- Facebook permissions, 373
- Facebook requests, 375-376
- login and session management, 374

**Uri.fromFile() method**, 331

**URIs (Uniform Resource Identifiers)**, 227-228, 331-332

**Uri.withAppendedPath() method**, 275

**user gestures**, 394

**user interface**. *See also views*

- action bar, 120
  - drop-down navigation, 120-121
  - tab navigation, 122-124
  - when to use, 124-125
- adapters, 104-105
- AsyncTask class, 108-109
- AutoCompleteTextView, 106
- demo app, setting up, 95-97
- dialogs
  - alert dialogs, 151-154
  - closing, 144
  - date picker dialogs, 146-149
  - displaying, 142-143
  - explained, 141-142
  - opening, 144
  - ProgressDialog, 146
  - time picker dialogs, 149-150
- fragments
  - with action bars, 134-136
  - AdapterViewFlipper, 166-167
  - displaying dynamically, 129-131
  - displaying with layouts, 128-129
  - explained, 127-128
  - fragment and activity interaction, 136-138
  - galleries, 164-166
  - grid views, 162-163
  - ListFragments, 157-161
  - navigating between, 132-134
  - support package, 131
- Hour1App, 13-14
- ImageViews, 110
- input controls, 98
  - buttons, 100-103
  - EditText, 98-100
  - TextView, 98-100
- layouts
  - child views, 86-88
  - designing, 80
  - editing, 81
  - explained, 79-80
  - for fragments, 128-129

- FrameLayout control, 82, 89
- Layout Resource Editor, 80
- LinearLayout control, 82-84
- RelativeLayout control, 82, 89-91
- TableLayout control, 82
  - using programmatically, 81
- ListFragments
  - creating, 158-159
  - customizing, 159-161
- options menus, 113
  - adding icons to, 117
  - displaying, 114-119
  - responding to, 119-120
  - when to use, 124-125
- paging controls, 167-168
- ProgressBars, 107, 109-110
- SeekBar, 107, 110
- spinner control, 105-106
  - getting data from, 106
  - setting up, 105
- user gestures, 394
- user preferences, 174**
  - CheckBoxPreference, 175-177
  - EditTextPreference, 179
  - ListPreference, 177-179
  - preference activities, 183
  - PreferencesFragment, creating, 174-175
  - reading, 182
  - titles, adding, 180-181
- <uses-feature>, 401**
- <uses-feature> element, 67**

## V

- versions of Android**
  - action bar, 68-70
  - fragments, 68-70
  - strategy for device support, 72
  - support library, 70-71
  - table of, 68

- video**
  - capturing with intents, 336-338
  - CustomVideoActivity.java, 345
  - loading, 358
  - showing, 358-359
  - starting, pausing, and positioning, 358

### VideoViews

- layout, 357
- listeners, 358-359
- loading video, 358
- showing video, 358-359
- starting, pausing, and positioning video, 358

### ViewFlipper, 168

- ViewGroup controls, customizing, 395**

### ViewHolder pattern, 255-256

- viewing source code, 10-11**

### ViewPagerIndicator, 391

### ViewPager, 168

### ViewPropertyAnimator, 388-390

### views

- AdapterViewFlipper, 166-167
- AutoCompleteTextView, 106
- child views, 86-88
- custom view controls, 395
- custom views in alert dialogs, 154
- flipping, 166-167
- grid views, 162-163
- HorizontalScrollView, 168
- ImageView, 110
- ImageViews, 347
  - alpha level, 353
  - displaying images, 348
  - rotating images, 351-352
  - ScaleTypes in, 348-351
- starting, pausing, and positioning video, 358
- styles, 55-56
- SurfaceView, associating camera with, 339-340
- VideoViews
  - layout, 357
  - listeners, 358-359
  - loading video, 358
  - showing video, 358-359

- ViewFlipper, 168

- ViewPager, 168

- ViewPropertyAnimator, 388-390

- virtual devices. See AVD (Android Virtual Device)**

- visual editor, 97**

## W-X-Y-Z

- weight (child views), 88**

- Wharton, Jake, 391**

- wireframe diagrams, 260-262**

- WRITE\_EXTERNAL\_STORAGE permission, 338**

- XML layout, 10-11**

- XMLPullParser, 202**