

# Android UI Fundamentals

DEVELOP AND DESIGN



Jason Ostrander

# Android UI Fundamentals

**DEVELOP AND DESIGN**

**Jason Ostrander**



## **Android UI Fundamentals: Develop and Design**

Jason Ostrander

### **Peachpit Press**

1249 Eighth Street  
Berkeley, CA 94710  
510/524-2178  
510/524-2221 (fax)

Find us on the Web at [www.peachpit.com](http://www.peachpit.com)  
To report errors, please send a note to [errata@peachpit.com](mailto:errata@peachpit.com)  
Peachpit Press is a division of Pearson Education.  
Copyright © 2012 by Jason Ostrander

Editor: Clifford Colby  
Development editor: Robyn Thomas  
Production editor: Myrna Vladic  
Copyeditor: Scout Festa  
Technical editor: Jason LeBrun  
Cover design: Aren Howell Straiger  
Interior design: Mimi Heft  
Compositor: Danielle Foster  
Indexer: Valerie Haynes Perry

### **Notice of Rights**

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact [permissions@peachpit.com](mailto:permissions@peachpit.com).

### **Notice of Liability**

The information in this book is distributed on an “As Is” basis without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

### **Trademarks**

Android is a trademark of Google Inc., registered in the United States and other countries. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN 13: 978-0-321-81458-6  
ISBN 10: 0-321-81458-4

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

*To my lovely wife, Susan,  
who tirelessly supports me in all of my adventures.*

## ACKNOWLEDGMENTS

I could write an entire book thanking people for their help along the way. Instead, I'll have to settle for this short paragraph:

Thanks to Chris H. for pushing me to consider writing a book and giving me endless encouragement and support. To Cliff C. for giving me the chance to write this book. To Robyn T. for keeping me on schedule despite my best efforts. To JBL for fixing my code and rocking a mean bass guitar. To Scout F. and Myrna V. for working tirelessly when I was late getting chapters to them. To Lucas D. and Rob S. for reading early chapters and giving me valuable feedback. To the entire team at doubleTwist for their dedication to making great Android software. To the Android team at Google for creating a great platform. To my family for their continuing support despite my dropping off the face of the earth. To Peachpit for giving me the opportunity to write this for you. And to you, the reader, for giving me the chance to teach you in whatever small way I can.

## BIO

Jason Ostrander is a web and mobile software developer working at Silicon Valley startup doubleTwist, where he makes syncing media to Android phones simple. Prior to that, he solved networking problems at energy management startup Sentilla and defense company Northrop Grumman. Jason holds an MS in electrical engineering from UCLA. He lives with his wife in San Francisco's Mission District, where he spends his time searching for the perfect chile relleno.

# CONTENTS

Introduction .....	viii
Welcome to Android .....	x

---

## PART 1 BASIC ANDROID UI

<b>CHAPTER 1</b>	<b>GETTING STARTED .....</b>	<b>2</b>
	Hello World .....	4
	Basic Structure of an Android App .....	9
	Android UI Basics .....	14
	Tools .....	22
	Wrapping Up .....	32
<b>CHAPTER 2</b>	<b>CREATING YOUR FIRST APPLICATION .....</b>	<b>34</b>
	Creating an App .....	36
	Getting Started with Android Views .....	37
	Arranging Views .....	41
	Displaying a List .....	52
	Understanding Activities .....	57
	Preventing ANRs .....	64
	Finishing the TimeTracker App .....	71
	Wrapping Up .....	73
<b>CHAPTER 3</b>	<b>GOING FURTHER .....</b>	<b>74</b>
	Supporting Multiple Screen Sizes .....	76
	Handling Notifications .....	84
	Handling Events .....	92
	Creating Menus .....	96
	Implementing the Time Tracker .....	102
	Wrapping Up .....	107

---

	<b>PART 2 THE VIEW FRAMEWORK</b>	
<b>CHAPTER 4</b>	<b>BASIC VIEWS</b> .....	<b>110</b>
	Creating a Basic Form .....	112
	Displaying Images .....	124
	Creating Maps and Displaying Websites .....	130
	Wrapping Up .....	136
<b>CHAPTER 5</b>	<b>REUSABLE UI</b> .....	<b>138</b>
	Abstracting Your Layouts .....	140
	Abstracting Styles and Themes .....	148
	Using Fragments .....	153
	Wrapping Up .....	162
<b>CHAPTER 6</b>	<b>NAVIGATION AND DATA LOADING</b> .....	<b>164</b>
	Introducing the Action Bar .....	166
	Navigating Your App .....	172
	Loading Data into Views .....	181
	Wrapping Up .....	186
<b>CHAPTER 7</b>	<b>ANDROID WIDGETS</b> .....	<b>188</b>
	Creating a Basic Widget .....	190
	Creating a Collection Widget .....	206
	Wrapping Up .....	211

	<b>PART 3 ADVANCED UI DEVELOPMENT</b>	
<b>CHAPTER 8</b>	<b>HANDLING GESTURES</b> .....	<b>214</b>
	Listening to Touch Events .....	216
	Responding to Gestures .....	224
	Wrapping Up .....	229
<b>CHAPTER 9</b>	<b>ANIMATION</b> .....	<b>230</b>
	Creating Drawable Animations .....	232
	Creating View Animations .....	235
	Creating Property Animations .....	246
	Wrapping Up .....	255
<b>CHAPTER 10</b>	<b>CREATING CUSTOM VIEWS</b> .....	<b>256</b>
	Understanding How Android Draws Views .....	258
	Creating a Custom View .....	259
	Adding Custom Attributes to Your Custom Views .....	267
	Creating Compound Components .....	274
	Wrapping Up .....	279
<b>CHAPTER 11</b>	<b>CREATING ADVANCED GRAPHICS</b> .....	<b>280</b>
	Using Canvas .....	282
	Using RenderScript .....	289
	Using OpenGL .....	294
	Wrapping Up .....	301
<b>CHAPTER 12</b>	<b>LOCALIZATION AND ACCESSIBILITY</b> .....	<b>302</b>
	Making Your App Available in Multiple Languages .....	304
	Making Your App Accessible .....	309
	Wrapping Up .....	315
	Index .....	316



## INTRODUCTION

---

There is a revolution happening in the technology industry. Touchscreen interfaces, combined with low-cost and ubiquitous smartphones, have created a perfect storm for disruptive innovation. Android is at the forefront of this change, bringing a free and open-source platform on which developers can create the next generation of applications. With free development tools and an open market, anyone can develop applications that reach a worldwide market. But why choose to develop for Android?

Android now runs on the majority of smartphones in the United States. And it's quickly expanding into new markets and device types. The last year has seen the introduction of hundreds of Android-based tablets, including the hit Kindle Fire. Google has ported Android to TVs with its Google TV platform, and many manufacturers are beginning to ship TVs with Android built in. Boeing has selected Android as the entertainment platform for its new Dreamliner jet. Ford is integrating Android into its in-dash SYNC entertainment system. And Android is quickly gaining traction in the developing world, especially in Africa, where the need for low-cost handsets is greatest.

Yet for all of the platform's promise, the majority of Android applications still lack the visual polish of their iOS counterparts. This book aims to address that issue by providing developers with a solid foundation for building app UIs. It will cover the basics of UI development on Android, teach best practices for creating flexible layouts, and give you tips on how to optimize your UI for smooth, fluid performance. I created this book in the hope that it will help developers to create beautiful applications.

Who am I? I've been developing software professionally for almost ten years, and I've focused on embedded and mobile software for the last five. In my day job, I work for one of the top Android development companies and write code that millions of people use every day.

Android development can be difficult at times, and the challenges of supporting such a diversity of devices can be daunting. But with a good idea, a solid understanding of the framework, and a little persistence, anyone can create a great app that is used by millions of people.

I hope you'll enjoy reading this book as much as I enjoyed writing it for you.

---

## WHO THIS BOOK IS FOR

This book is aimed at beginning Android developers who are interested in creating great user interfaces. You are expected to know basic Java programming and XML syntax. The focus of this book is on UI. While you don't need to have experience writing Android software, many basic Android concepts are only described in passing. It will help you to have a rudimentary knowledge of Android development.

## WHO THIS BOOK IS NOT FOR

This book is not a general introduction to programming for Android. While it is intended for beginning Android developers, the focus is on user interface tools and programming. In particular, this book will not cover basic Android concepts such as intents, services, or content providers. Further, this book will not be an introduction to the Java programming language or to XML. You should know how to program and how to read XML.

## HOW YOU WILL LEARN

Throughout this book, you'll learn by creating an actual app, a simple time tracker. Each chapter includes detailed examples of real Android code that you will compile and run. All code for the book can be found at the book's website: [www.peachpit.com/androiduifundamentals](http://www.peachpit.com/androiduifundamentals).

## WHAT YOU WILL LEARN

You'll learn how to create user interfaces for Android applications. From the most basic concepts, like activities and views, all the way to advanced graphics using RenderScript, this book covers everything you will use to build the user interface of your apps.

## A NOTE ABOUT ANDROID VERSIONS

This book was written to Android version 4 APIs and best practices, but it is compatible back to Android version 2.2. When relevant, notes and tips are included to indicate when an API is deprecated or no longer appropriate. The Android compatibility library, a package of classes that back-port several newer features, will be used throughout the book.

## WELCOME TO ANDROID

Throughout this book, you'll be writing your code using the Eclipse integrated development environment (IDE). You'll need to install the Android software development kit (SDK), along with the Android Developer Tools (ADT) plugin. This setup includes several other utilities to help you develop and test your application. Aside from the SDK, none of these tools are required, but they will make your application development easier.

### THE TOOLS

The following tools are used throughout this book to build, test, and analyze your applications.



#### ANDROID SDK

The Android SDK is required to build and deploy Android applications. The SDK contains the tools you'll use to test and debug your application. It also contains tools for creating flexible layouts. You can download the Android SDK at <http://developer.android.com/>.



#### ECLIPSE

Eclipse is the recommended IDE for Android development and is the only IDE officially supported by Google. Google publishes a plugin called Android Developer Tools that integrates with Eclipse and provides features like a drag-and-drop interface builder. You are not required to use Eclipse, as the Android SDK fully supports command-line development. Throughout this book, however, it is assumed you are using Eclipse. You can download Eclipse at [www.eclipse.org](http://www.eclipse.org).



### ANDROID SDK MANAGER

The Android SDK Manager is used to download and install the Android SDK. You will also use the SDK Manager to install add-on features like sample code, third-party emulators, and the compatibility library. The Android SDK Manager can also be used to create and launch emulated Android devices, called Android Virtual Devices. The Android SDK Manager can be found in the SDK `tools/` directory as `android`.



### HIERARCHY VIEWER

This tool displays a graphical representation of your layout hierarchy and can help you debug layout performance issues by spotting overly complex layouts. It's also a good tool for understanding how Android layout works. You can find this tool in the SDK `tools/` directory as `hierarchyviewer`.



### DDMS

The Dalvik Debug Monitor Server (DDMS) is used to debug application performance issues. It provides Java heap usage, running thread counts, and object allocation tracking. You also use it to take screen shots. The DDMS tool is built into Eclipse through the ADT or can be run standalone from the `tools/` directory of the SDK.

*This page intentionally left blank*

---

# 4

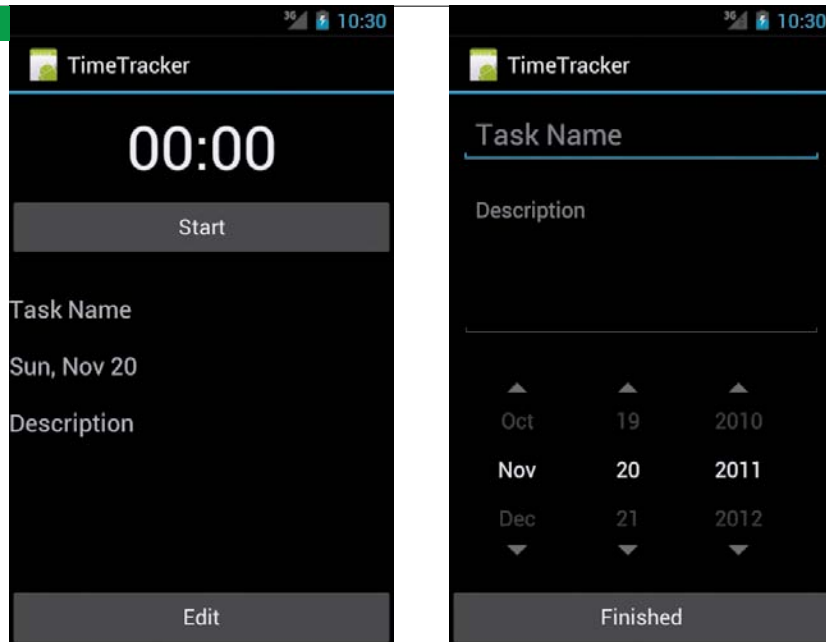
# BASIC VIEWS



The most basic element of Android user interfaces is the View class. A view represents an area of the screen. Buttons, lists, webpages, and even empty spaces are represented by views. Android contains a rich array of pre-built View classes that provide much of the functionality you will need. When the built-in views aren't enough, it's possible to create special views that are just right for your application. In this chapter, you will learn about the basic view types you can use to build your layout, discover how to load and display images, and explore the more advanced views available in Android: MapView and WebView.

## CREATING A BASIC FORM

**FIGURE 4.1** The TimeTracker app will have task detail and task edit screens.

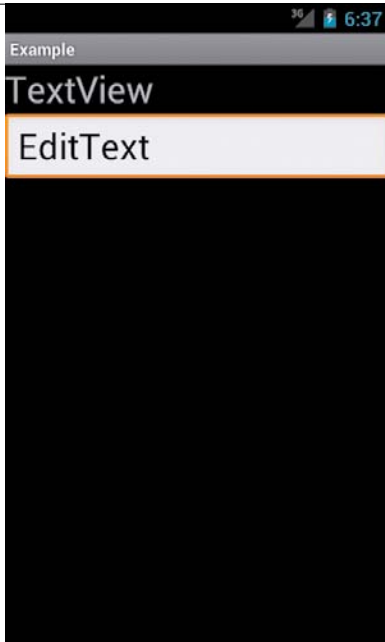


The TimeTracker app looks pretty good so far, but it's time to add more than just a list of times. In this chapter, you'll add some text entry forms and split the app into multiple activities. When you're finished, you'll have something that looks like **Figure 4.1**. This section will cover the basic widgets you see in the image, as well as how to arrange them.

### TEXTVIEW AND EDITTEXT

The most basic view available on Android is the `TextView`, which allocates an area of the screen to display text. You will use this view a lot in your layouts. An `EditText` is a `TextView` that is configured to allow the user to edit the text inside it (**Figure 4.2**). Tapping an `EditText` will display a cursor and the device software keyboard, allowing the user to enter new text or edit the existing text. The `TextView` has optional attributes such as size, font, and color that allow you to change the appearance of the text.





**FIGURE 4.2** A TextView and an EditText

### CREATING THE TEXTVIEW

To create the new UI for the TimeTracker app, you'll need to create two new layouts: `task_detail.xml` and `edit_task.xml`. They will look very similar, but `edit_task.xml` will use `EditText` instead of `TextView`. Here is the XML for `task_detail.xml`:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/counter"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
        android:gravity="center"
        android:padding="10dp"
        android:text="@string/sample_time"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textSize="50sp" >
</TextView>
<Button
    android:id="@+id/start_stop"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="30dp"
    android:text="@string/start" />
<TextView
    android:id="@+id/task_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:text="@string/task_name"
    android:textSize="20dp" >
</TextView>
<TextView
    android:id="@+id/task_date"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="20dp"
    android:text="@string/date_select"
    android:textSize="20dp" />
```

```
<TextView
    android:id="@+id/task_desc"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_marginBottom="20dp"
    android:layout_weight="1"
    android:text="@string/description"
    android:textSize="20dp" />
</LinearLayout>
```

This XML layout keeps the counter and the Start/Stop button from Chapter 2, but the task list is replaced with the new task detail fields. Note the use of `layout_weight` on the description to fill the entire display.

#### SIMPLIFYING TEXT ENTRY

In addition to general text entry, you will probably want your users to enter textual data in a particular format. Data such as email addresses, phone numbers, and passwords are particularly common on a mobile device. With a hardware keyboard, the user just enters data normally, but because Android devices have a software keyboard, the keys can be changed to make entry of certain data types easier. For example, if you have a field that accepts only numerical data, the keyboard will display just the number pad.

**NOTE:** In addition to changing the input, Android supports changing the entire software input editor, or IME. The typical IME is a software keyboard, but Android also supports IMEs like voice input, handwriting recognition, or even Palm OS-inspired graffiti. While this is not something you control with your app, you can give hints about the actions that should be taken when inputting data into forms; those hints will then be used to select the appropriate IME.



**FIGURE 4.3** The keyboard displayed when the `inputType` of an `EditText` is set to phone



The `inputType` attribute of your `EditText` class is a simple bit mask that defines the type of data you expect the user to enter. The system can then display an appropriate keyboard type. You can combine `EditText` flags (attributes) so that the system creates a targeted input keyboard. For example, the following `EditText` attributes will make the keyboard a number pad for easy entry of phone numbers (**Figure 4.3**):

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="phone" />
```

Along with changing the keyboard, you can use `inputType` to change the behavior of the `EditText`; for example, use flags like `textCapSentences` and `textAutoCorrect` to add capitalization and autocorrection to what the user types. In addition to configuring the input options, you can use an IME option to set the text for the Enter button, which appears in the lower-right corner of the stock Android keyboard: Use the `imeOptions` attribute to select `actionGo`, `actionSearch`, `actionSend`, `actionNext`,

or `actionDone` to give the user a visual indication of what action will be taken when they are finished entering text.

Now you can create the content of the `edit_task.xml` layout. Create the file, and add the following XML:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/task_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/task_name"
        android:layout_margin="10dp"
        android:textSize="24dp" >
    </EditText>
    <EditText
        android:id="@+id/description"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_margin="10dp"
        android:hint="@string/description"
        android:gravity="top|left" />
    <DatePicker
        android:id="@+id/datePicker1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_gravity="center_horizontal"
        android:calendarViewShown="false"
        android:layout_margin="10dp" />
</LinearLayout>
```

Here you're using the `android:hint` attribute rather than `android:text`. This displays the desired preset text but removes it as soon as the user starts typing a value into the field. This `edit_task.xml` layout also uses the `DatePicker` view to make date entry easier.

## BUTTONS

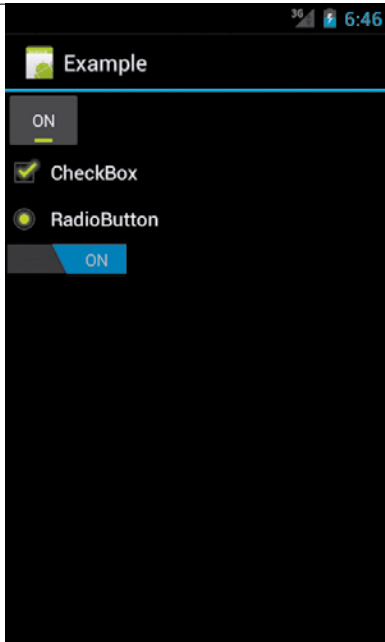
You've already used buttons to build the current `TimeTracker` UI. Buttons are simply `TextViews` that have a special background image—this background is actually an XML file that lists the images that should be used for the different button states (normal, hovered, focused, and pressed). This type of XML resource is called a *state list* resource, and you'll learn more about creating it later in this chapter.

1. Add a Finished button to the `edit_task.xml` layout:

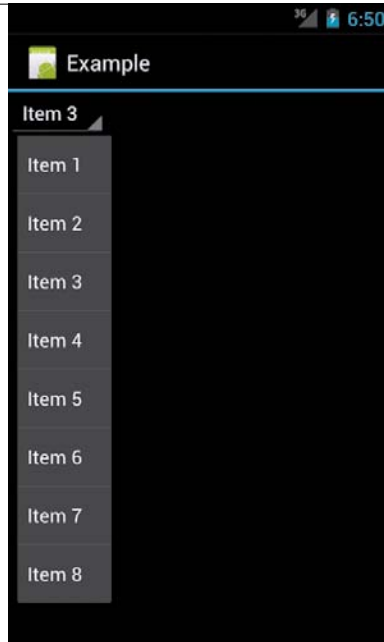
```
<Button
    android:id="@+id/finished"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/finished" >
</Button>
```

2. Add an Edit button to the `task_list.xml` layout:

```
<Button
    android:id="@+id/edit"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/edit" >
</Button>
```



**FIGURE 4.4** Boolean buttons on Android 4.0



**FIGURE 4.5** A spinner on Android 4.0

## BOOLEAN BUTTONS

Buttons are convenient for indicating on/off states. Android has a number of views, including toggle buttons, checkboxes, and radio buttons, that subclass the `Button` class and present a toggle between a true value and a false value. In addition, Android 4.0 introduced an option called the switch. **Figure 4.4** shows all these options for the 4.0 release of Android.

## SPINNERS


A spinner looks like a button and displays a list of choices when pressed. **Figure 4.5** shows an example of a spinner choice list. The options presented by a spinner can be specified using the XML `android:entries` attribute, or you can use a data adapter to load entries programmatically (you'll learn more about loading entries into views via data adapters in Chapter 6).

## MANAGING SETTINGS

Often, you will want to give users the ability to change the general options of your app through settings screens. It's not necessary to create a form, because Android includes a set of classes designed to create settings screens. The basic class is the `Preference`, and there are several different preference forms, mimicking the standard UI form widgets. The user's preferences will be saved to a key-value store that is local to your app.

Prior to Android 3.0 (Honeycomb), you would use a `PreferenceActivity` class for displaying application preferences. Honeycomb and later releases use the new `PreferenceFragment` class to handle settings preferences. However, this class is not available in the compatibility library, so you will need to continue using the `PreferenceActivity` class for applications that are designed to run on Android 2.3 and earlier.

## SCROLLVIEW



Adding entry fields to a form is simple, but what happens if you cannot fit all the views on one screen? In these cases, it's often useful to allow scrolling in order to fit more elements in a single activity. To achieve this effect, you need to wrap your views in a `ScrollView` container. A `ScrollView` allows you to create a view that is larger than the physical screen on a device and scroll it to reveal the full contents. `ScrollView` is actually a subclass of `FrameLayout`, but it adds the ability to scroll its content. You typically place another layout container inside the `ScrollView` to arrange the child views.

**TIP:** You should never use a `ListView` inside a `ScrollView`.

The behavior will be erratic and unpleasant to the user. If you find yourself wanting to use both, consider redesigning your app to use one or the other.



---

Since you want the user to enter an arbitrary amount of description text in the time tracker, you'll want to use a `ScrollView` so they can see it all. Wrap the existing `LinearLayout` contents in a `ScrollView`:

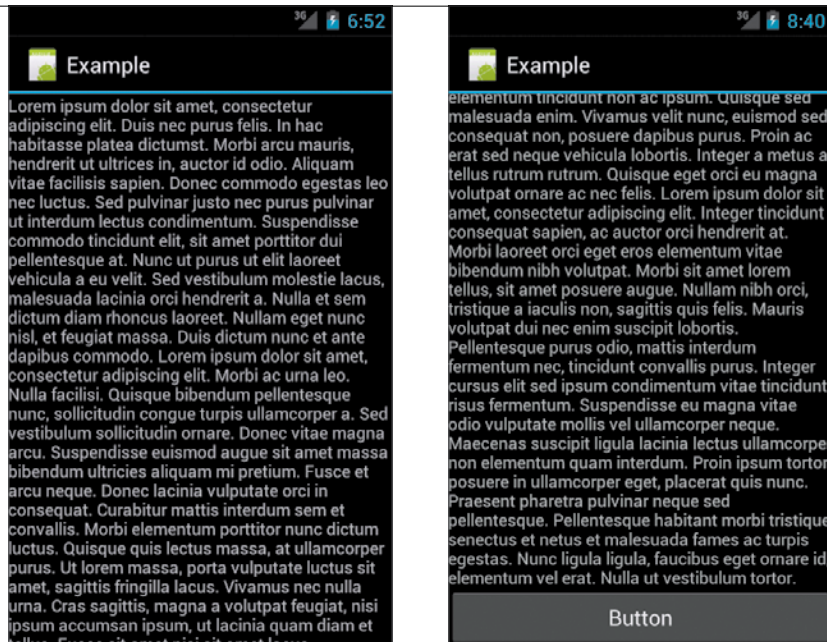
```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fillViewport="true" >
    <LinearLayout>
    <!-- Rest of code here -->
    </LinearLayout>
</ScrollView>
```

This code should be self-explanatory by now. The `ScrollView` simply wraps the `LinearLayout`, which contains the text and buttons you have already created. Notice the `android:fillViewport` attribute? This prevents some odd behavior, which you'll learn about next.

#### THE FILLVIEWPORT ATTRIBUTE

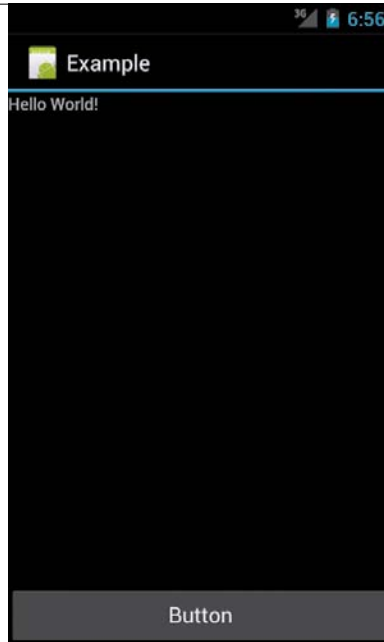
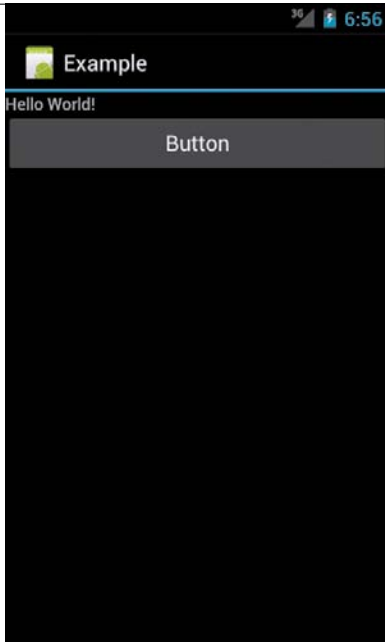
A common issue you may experience with `ScrollView` is its interaction with child views that are smaller than the display. When the child view is larger than the display, the `ScrollView` behaves as expected, allowing you to scroll to see the full view. However, when the child view is smaller than the display, the `ScrollView` will automatically shrink itself to match the size of its content. The proper way to handle this is to use the `fillViewport` attribute, which will cause the child views of a `ScrollView` to expand to the size of the display, if necessary; if they are already larger than the display, nothing happens. A simple example will demonstrate.

**FIGURE 4.6** The desired ScrollView result, with a long block of text scrolling to reveal a button



A frequent task is displaying a block of text with a button at the bottom (such as in a license agreement to which a user must agree). **Figure 4.6** shows the desired result: a long block of text that scrolls to reveal a button. When the text is smaller than a single screen, the naive implementation of `ScrollView` results in **Figure 4.7**—the button should still be pinned to the bottom of the screen but is instead directly below the text. The `ScrollView` only takes up as much space as its content. To fix this, set the `fillViewport` attribute to `true`. Here is the code to correctly implement scrolling for any size of text, resulting in **Figure 4.8**.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:fillViewport="true" >
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```



**FIGURE 4.7** The ScrollView result if the fillViewport attribute is not set to true

**FIGURE 4.8** Because the android:fillViewport attribute was used, the button is now correctly pinned to the bottom of the screen.

```
android:orientation="vertical" >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="Odp"
    android:layout_weight="1.0"
    android:text="@string/hello" />
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button" />
</LinearLayout>
</ScrollView>
```

Try using ScrollView with and without the fillViewport attribute to see how its behavior changes.

## DISPLAYING IMAGES

Android phones feature large, high-resolution displays that are perfect for displaying images in your application. Images are an important way of conveying information to your users without explicitly stating it. Typically, images are displayed using the built-in image view. This view takes care of the loading and optimizing of the image, freeing you to focus on app-specific details like the layout and content. Unless you need special optimizations for your application, you should take advantage of the built-in image view whenever possible.

### IMAGEVIEW AND RESOURCES

The simplest way to display an image is to declare an `ImageView` in your layout file and set its source to a resource in your project. Image resources are placed in the `/res/drawable` folders. This example will display an image named “icon”:

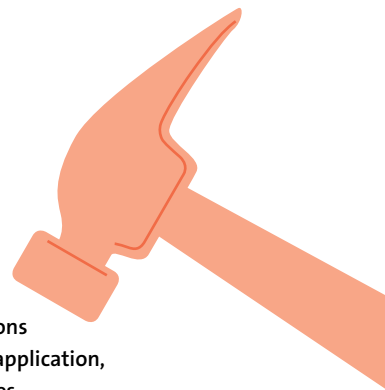
```
<ImageView
    android:id="@+id/image"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="center"
    android:src="@drawable/icon" />
```

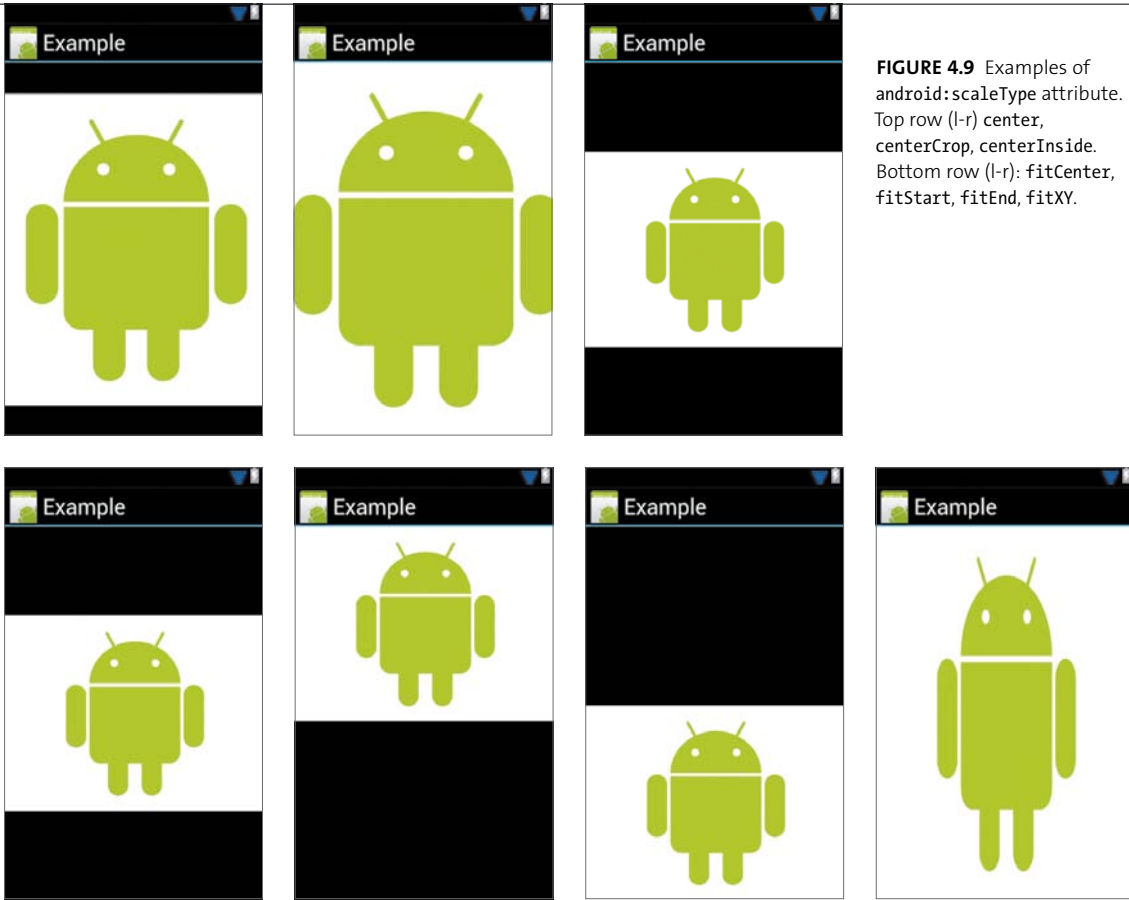
The `ImageView` handles all the loading and scaling of the image for you. Note the `scaleType` attribute? This defines how the images will be scaled to fit in your layout. In the example, using `scale type center`, the image will be displayed at its native resolution and centered in the view, regardless of how much space the view consumes. Other scaling options fit the image to the dimensions of the image view or scale the image based on the width and height of the device. **Table 4.1** lists the scale type options and how they alter the image.

**TABLE 4.1** `ImageView` Scale Types

SCALE TYPE	DESCRIPTION
<code>center</code>	Displays the image centered in the view with no scaling.
<code>centerCrop</code>	Scales the image such that both the <i>x</i> and <i>y</i> dimensions are greater than or equal to the view, while maintaining the image aspect ratio; crops any part of the image that exceeds the size of the view; centers the image in the view.
<code>centerInside</code>	Scales the image to fit inside the view, while maintaining the image aspect ratio. If the image is already smaller than the view, then this is the same as <code>center</code> .
<code>fitCenter</code>	Scales the image to fit inside the view, while maintaining the image aspect ratio. At least one axis will exactly match the view, and the result is centered inside the view.
<code>fitStart</code>	Same as <code>fitCenter</code> but aligned to the top left of the view.
<code>fitEnd</code>	Same as <code>fitCenter</code> but aligned to the bottom right of the view.
<code>fitXY</code>	Scales the <i>x</i> and <i>y</i> dimensions to exactly match the view size; does not maintain the image aspect ratio.
<code>matrix</code>	Scales the image using a supplied <code>Matrix</code> class. The matrix can be supplied using the <code>setImageMatrix</code> method. A <code>Matrix</code> class can be used to apply transformations such as rotations to an image.

**TIP:** The `fitXY` scale type allows you to set the exact size of the image in your layout. However, be mindful of potential distortions of the image due to scaling. If you're creating a photo-viewing application, you will probably want to use the `center` or `fitCenter` scale types.





**FIGURE 4.9** Examples of `android:scaleType` attribute. Top row (l-r) `center`, `centerCrop`, `centerInside`. Bottom row (l-r): `fitCenter`, `fitStart`, `fitEnd`, `fitXY`.

Figure 4.9 shows examples of the scale types. Using the correct scale type is important if you want to properly display images.

## BITMAPS

Images used in your application are stored in the `/res/drawable` folders. These folders follow the device-configuration naming scheme to provide different images for different devices. Typically, you will create four different versions of each image and place them in the following folders: `drawable-ldpi`, `drawable-mdpi`, `drawable-hdpi`, and `drawable-xhdpi`. These represent the increasing resolutions of each device, and appropriately sized images should be placed in each. Use the same filename for each of the different versions, and then when you specify the drawable name, the Android resources manager will choose the image from the appropriate folder.

It's not always necessary to create an image for a particular resolution; Android will display whatever image is the best match. In general, Android will prefer scaling an image down in size so that images are always crisp and not blurred. By default, you should create hdpi-resolution images. However, you should strive to create resources for all resolutions to prevent unnecessary hardware scaling, which slows down the drawing of your UI. Once your image resources are placed in the `res/drawable` folders, you can reference them the same way you reference your layout files: via the `R.java` file.

**NOTE:** Image resources in your project should be in one of three formats: PNG (preferred), JPEG (acceptable), and GIF (discouraged). Of course, 9-patch images are also accepted.

Including images in the `res/drawable` folders is a simple way of adding images to your app. However, it's also possible to create images at runtime and add them to your layout. For example, you may want to download an image from the Internet and display it to the user. To do this, you create a `Bitmap` object to encapsulate the image, and then load it into your UI. The `Bitmap` class is simply an object that references a bitmap image. You can use a `BitmapFactory` to create a bitmap image from any source: a resource in your app, a file, or even an arbitrary `InputStream`. A bitmap can then be loaded into an image view by calling `setImageBitmap`. Here is an example:

```
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image);
ImageView iv = (ImageView) findViewById(R.id.image);
iv.setImageBitmap(bitmap);
```



## DRAWABLES

Not all graphics need to be images—Android also lets you create graphics by using XML or writing custom drawing code. You’ll learn more about creating custom graphics using Canvas and other classes in Chapter 11. To create graphics using XML, you use the Drawable class. A drawable represents something that can be drawn on the screen. This can be an image, an XML resource, or a custom class. The Drawable class is the general abstraction for representing all of these in your UI.

The Android framework makes extensive use of drawables for all the built-in UI views. One of the most common is the Button class, which uses an XML file to define the possible states a button can have. Here is an example XML file for Button:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_pressed="true"
        android:drawable="@drawable/button_pressed" />
  <item android:state_focused="true"
        android:drawable="@drawable/button_focused" />
  <item android:state_hovered="true"
        android:drawable="@drawable/button_hovered" />
  <item android:drawable="@drawable/button_normal" />
</selector>
```

This is called a `StateListDrawable`. It defines a set of drawables associated with different states. In this example, there are four possible states the button can be in: normal, hovered, focused, and pressed. Each item in the `StateListDrawable` defines a drawable that will be displayed when the button is in the specified state. In this case, the `android:drawable` attribute references an actual image drawable. The `StateListDrawable` does not select the best matching item, but rather selects the first item that meets the criteria for the current state. It performs this search from top to bottom, so the order in which you place each item is important. Using different drawables for button states provides feedback to the user when they are interacting with the UI.



---

There are more options than just defining states for a drawable. There are formats that create simple transformations of an existing bitmap or add padding and dithering to an image. You can combine several bitmaps to create a composite image. Or you can use XML to actually draw a shape using the `ShapeDrawable` class. You can add gradients, shadows, and rounded corners. The full range of XML drawable options is outside the scope of this book, but you should familiarize yourself with the available options. If you find yourself contemplating creating custom graphics to achieve the effects you want, consider using a drawable resource that may already be available.

In addition to displaying images using drawables and bitmaps, you have the option to create custom graphics using classes like `Canvas`, `SurfaceView`, and `TextureView`. You'll learn more about this in Chapter 11.

**NOTE:** Drawing images into a view uses the system's standard drawing process. In Android versions earlier than 3.0, this process is not fully hardware accelerated. Be aware that graphics-intensive applications using this process will not perform well on older versions of Android.

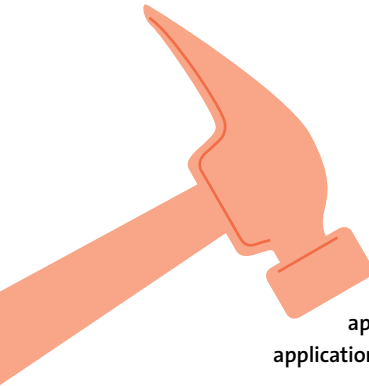


## CREATING MAPS AND DISPLAYING WEBSITES

The typical Android device ships with a built-in GPS receiver and an always-on network connection. This provides tremendous opportunities for developers to leverage these features and create compelling location-aware applications. Android devices include access to Google's mapping technology, which you can use to add full-fledged navigation to your app. And the built-in Webkit browser gives you the power to create your own web-browsing applications. The next sections cover the basics of using these advanced views.

### MAPVIEW

Unlike other views and classes in Android, maps are not part of the core library. They are provided by Google and are available to any application running on an Android-compatible device. Notably, this does not include devices that do not conform to the Android Compatibility Definition, such as the Kindle Fire. You will be unable to use Google Maps on those devices. However, most devices meet the Android specifications and support Google Maps integration.

An illustration of a hammer with a wooden handle and a metal head, positioned diagonally on the left side of the page. The hammer is orange and red in color.

**TIP:** Make sure you properly declare your permissions in the application manifest file. If you want to use location features in your application, you will need to request the location permissions in your app.

You can set up your project to use maps as follows:

1. Visit the Google APIs site (<http://code.google.com/android/add-ons/google-apis/>), and register for a map key. Map views are provided as part of the `com.google.android.maps` package, and you will need to register for a Google Maps API key in order to use this package.
2. Using the Android SDK Manager, download the Google APIs version of the Android SDK that you intend to support. You can use this SDK to create a new AVD image that supports MapView. Make sure you select a Google APIs target for your image.

3. Declare that your application requires the external Google Maps library to run by adding this to your manifest under the `<application>` element:

```
<uses-library android:name="com.google.android.maps" />
```

4. Google Maps requires a network connection, so you need to add the `android.permission.INTERNET` permission to your manifest:

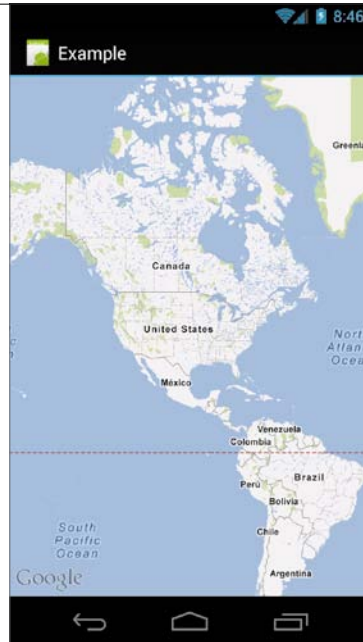
```
<uses-permission android:name="android.permission.INTERNET" />
```

With those tweaks, you can use maps in your application. You add a map view to your layout like you would add any other view:

```
<com.google.android.maps.MapView  
    android:id="@+id/mapview"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:apiKey="Your Maps API Key"  
    android:clickable="true" />
```

Note that the element name highlighted in the code is the full package name—anytime you use a custom view that is not part of the core Android library, you need to specify the full package name. You will need to declare the ID of the `MapView` as `mapview`. Also, there are two new attributes here. The first is the `apiKey` attribute, which is where you will place the Google Maps API key you get from Google. This enables you to use Google's mapping service. The second new attribute is the `clickable` setting. Setting this to `true` allows the user to tap and scroll on the `MapView` in your UI; setting it to `false` will prevent all interaction with the map.

**FIGURE 4.10** A MapView example.



To actually use a map view in your layout, your activity will need to extend `MapActivity`, which handles all the setup of the map view, and override the `isRouteDisplayed` method, which is required by the Google Maps license agreement and should return a Boolean that indicates whether there is active routing information displayed on the map (Figure 4.10).

**NOTE:** Because your activity must extend `MapActivity`, you cannot use fragments from the compatibility library and use a map view at the same time. For Android 3.0 and above, the fragment framework is built in to the `Activity` class, so this is not an issue.

---

## WEBVIEW

Android includes a Webkit-based HTML rendering engine backed by the V8 JavaScript interpreter. You can use these technologies in your own application by using the `WebView` class. A web view renders HTML from web URLs, files stored on the device, or arbitrary strings you create in your app. Android's `WebView` includes standard browser features like history, plugins, zooming controls, and JavaScript support. You can also enable advanced gestures like pinch to zoom, providing easy navigation on touchscreen devices.

Like the map view, the web view can be added to your application with a simple XML element:

```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

You will need to enable the `INTERNET` permission in your manifest for your web view to access online webpages. The web view does all downloading and rendering of webpages, and you won't need to extend any special activities or use a special ID. With a web view in your UI, loading a webpage is as simple as adding the following code:

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.loadUrl("http://www.google.com");
```

With that, you can display any webpage to the user in your custom UI layout. Note that the supplied content highlighted in the example is an actual webpage URL. It's also possible to load an arbitrary string containing HTML for display.

The web view defaults don't include JavaScript or Flash support. To enable that, you'll need to use a `WebSettings` object:

```
WebSettings webSettings = webView.getSettings();
webSettings.setJavaScriptEnabled(true);
webSettings.setPluginState(WebSettings.PluginState.ON);
```

---

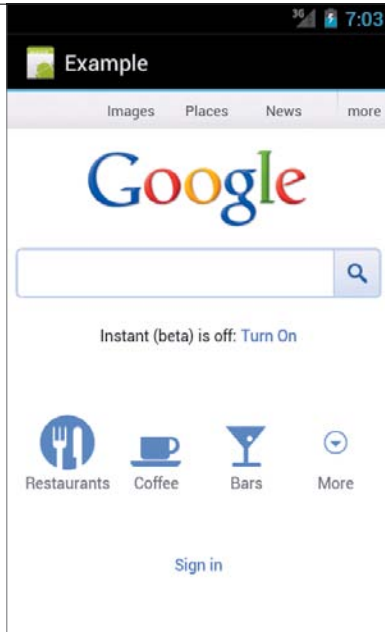
This enables JavaScript and plugins—including Flash, if it’s installed—in the web view. Adding zoom controls and pinch-to-zoom functionality is also simple:

```
webSettings.setSupportZoom(true);  
webSettings.setBuiltInZoomControls(true);
```

The first line indicates that the web view will support zooming its contents. The second line uses the web view’s built-in zoom controls for performing the zoom (this includes the tap-to-zoom and pinch-to-zoom functionality).

Finally, you will likely want to override the loading of new URLs in your web view. If you don’t do so, when the user taps on a new URL in the web view, the default browser will open to load the new link. To force the load to occur in your web view, add the following code:

```
webView.setWebViewClient(new WebViewClient() {  
    @Override  
    public boolean shouldOverrideUrlLoading(WebView view, String url) {  
        view.loadUrl(url);  
        return true;  
    }  
});
```



**FIGURE 4.11** The web view displaying Google's homepage

Here the URL loading behavior is overridden, and the new URL is loaded in the existing web view. Returning true will discontinue the propagation of the event up the view hierarchy and prevent the browser from opening. **Figure 4.11** shows the screen of this activity.

The web view allows you to present any HTML content to the user and provides an easy way to load pages from the Internet. You should take advantage of it whenever your application needs to display HTML content.

## WRAPPING UP

---

This chapter introduced the basic building blocks used to build a form on Android. You used these to refactor the TimeTracker app into a series of activities for displaying and entering tasks. You still need to save the data and display it, which we'll cover later in the book. In this chapter, you learned that

- Android provides a set of simple input widgets that you can use to build forms.
- Use the proper `android:scaleType` attribute when displaying an image using `ImageView`.
- With the `Drawable` class, you can create complex image types using only XML.
- Adding a map to your application is as simple as extending `MapActivity` and adding the map view to your layout.
- Android's Webkit-based `WebView` class allows you to display any HTML content.



*This page intentionally left blank*

# INDEX

@ symbol, using with resources, 17

## A

accelerated rendering

- disabling, 252
- enabling, 252

accessibility

- content description, 312–313
- contentDescription attribute, 314
- events, 313–314
- focus, 309–312
- guidelines, 314
- navigation, 309–312
- nextFocus attributes, 311–312
- screen reader, 314
- testing, 314
- tips, 314
- views, 314

action bar, 166. *See also* menus

- action items, 166–167
- action views, 169–170
- ActionProvider class, 170–171
- buttons, 167
- overflow menu, 169
- ShareActionProvider class, 171
- split, 168

action bar navigation, tabbed

- interface, 172–173

ActionBarSherlock library, 169

ActionBar.TabListener interface, 173

activities

- in back stack, 62
- callbacks, 59
- configuration changes, 63
- creating, 58
- creating and destroying, 19
- declaring, 57
- declaring intents, 57
- findViewById method, 60
- grouping into tasks, 61–63
- life cycle, 58–60
- manifest entry, 57
- as objects, 19
- onCreate method, 58
- onPause method, 58
- overriding onCreate method, 60
- Paused state, 58

popping off stack, 61

- Resumed state, 58
- Running state, 58
- saving current states of, 62
- setContentViews, 60
- states, 58
- Stopped state, 58
- XML layout file, 60

Activity class, 17–19

- callback structure, 18
- R.java file, 18–19
- setting views, 18

Adapter class

- using, 182–183
- ViewHolder pattern, 182–184

adapters, optimizing, 182–184

AdapterViewFlipper collection

- view, 206

alert dialog, 89

AlertDialog.Builder class, 90

Android API versions, declaring in manifest, 12

Android apps

- compatibility, 11
- folder structure, 9
- resources, 12–13
- responsiveness, 27

Android Asset Studio, 31

Android Device Chooser, 6–7

Android emulator, 6, 8, 26

Android manifest

- Android API versions, 12
- contents, 10
- hardware features, 11
- icons, 11
- labels, 11
- permissions, 11

android: prefix, using, 38

Android SDK, x

Android SDK Manager, xi

Android SDK Release 13, 4

Android Virtual Devices (AVDs), 26

- creating, 6–7
- emulated, 26
- graphics stack, 26

AndroidManifest.xml item, 9

android:maxSDKVersion, 12

android:minSDKVersion, 12

android:targetSDKVersion, 12

animated ball

creating, 232–234

ImageView, 233

stop() and start(), 233

animateDigit function, creating, 243–244

animation sets

- examples, 236–237
- ordering property, 251

AnimationListener, using, 239

animations. *See also*

- drawable animations;
- property animations;
- sliding animations;
- view animations
- clock-flipper, 240–245
- counter.xml, 241–242
- defining, 235–237
- digit.xml layout, 240
- fillAfter attribute, 239
- hardware acceleration, 252
- interpolators, 238
- startAnimation method, 238
- TextView, 238–240
- three-button layout, 310–311
- using, 238–239
- view transparency, 236–237

AnimatorSet object

- play method, 250
- using with property animations, 249–250

ANRs (Application Not Responding)

- background tasks, 66–70
- occurrence of, 64
- preventing, 64–70
- StrictMode, 64–65

API level declaration

- android:maxSDKVersion, 12
- android:minSDKVersion, 12
- android:targetSDKVersion, 12

app drawer

- contents, 14–15
- launcher for, 14

app launcher, using, 14

apps

- compatibility, 11
- folder structure, 9

- resources, 12–13
- responsiveness, 27
- AppWidgetProvider class, 197–200
  - onDeleted method, 197
  - onDisabled method, 197
  - onEnabled method, 197
  - onReceive method, 197
  - onUpdate method, 197, 201
- AppWidgetProviderInfo XML file,
  - 191–195, 209–210
  - autoAdvanceViewID, 194
  - configure, 194
  - icon, 194
  - initialLayout, 194
  - label, 194
  - minHeight, 194
  - minResizeHeight, 194
  - minResizeWidth, 194
  - minWidth, 194
  - previewImage, 194
  - resizeMode, 194
  - updatePeriodMillis, 194
- assets/ folder, 9
- AsyncTask class, 69–70. *See also* tasks
- attributes
  - adding to custom views, 267–273
  - AttributeSet object, 271
  - crosses, 270
  - CrossView, 267, 269–270, 273
  - declaring, 267–269
  - enums, 269
  - flags, 269
  - namespace, 269
  - predefined values, 268
  - rotating canvas, 272
  - using in code, 271–273
  - ViewGroup, 269
  - in XML, 269–270
- AttributeSet object, using, 271–273
- AVDs (Android Virtual Devices), 26
  - creating, 6–7
  - emulated, 26
  - graphics stack, 26

## B

- Back button, 19, 61–62
- back stack, 61–63 161

- background tasks. *See also*
  - TimeTracker app
- Activity.runOnUiThread
  - method, 68
- AsyncTask class, 69–70
- Handler class, 66
- handlers and message queues,
  - 66–68
- Looper class, 66
- message queue, 68
- resetTimer method, 68
- sendEmptyMessage method, 67
- stopTimer method, 68
- ball, animating, 232–234
- bitmaps, using, 127
- Boolean buttons, creating, 119
- build target, setting, 5
- Button
  - adding below TextView, 25
  - using, 128
  - view object, 37
- button states, 95
- buttons
  - creating for TimeTracker app, 118
  - in linear layout, 42–43
  - in relative layout, 46

## C

- callbacks, receiving, 64
- canvas
  - described, 22
  - drawing to SurfaceView, 282
  - implementing SurfaceView, 282
  - rotating, 272
  - TextureView class, 288
- circles, making for ball, 232–234
- click handling, avoiding anonymous
  - classes, 93
- clock-flipping animation
  - animateDigit function, 243–244
  - creating, 240–245
  - setting time, 242
  - sliding animations, 244–245
  - triggering, 242
- collection widget. *See also* views;
  - widgets
  - AdapterViewFlipper, 206

- GridView, 206
- layout, 206–207
- ListView, 206–207
- RemoveViewsFactory, 208–210
- service, 208–210
- StackView, 206
- colors
  - placing in res/values folder, 13
  - transparent, 194
- compound components
  - creating, 274–276
  - versus custom views, 274
  - optimizing layouts, 277–278
  - ToggleButton with EditText, 276
- configuration changes, handling, 62
- confirm dialog, creating, 90–91
- confirmation dialog, creating, 93
- constant values, placing, 13
- context menus
  - creating, 100–101
  - long pressing, 94
- counter.xml file, using with
  - animation, 241–242
- CrossView
  - changing color of, 273
  - rotating, 273
- CursorLoader example, 185
- custom attributes
  - adding to custom views, 267–273
  - AttributeSet object, 271
  - crosses, 270
  - CrossView, 267, 269–270, 273
  - declaring, 267–269
  - enums, 269
  - flags, 269
  - namespace, 269
  - predefined values, 268
  - rotating canvas, 272
  - using in code, 271–273
  - ViewGroup, 269
  - in XML, 269–270
- custom views. *See also* views
  - accessing resources, 263
  - activity for display of, 265
  - versus compound components, 274
  - creating, 259–260
  - CrossView, 265
  - custom attributes, 267–273

- custom views (continued)
  - drawLines method, 265
  - inner classes, 266
  - onDraw method, 263–266
  - onMeasure method, 260–262
  - Paint object, 263–266

## D

- data binding
  - SimpleAdapter, 181
  - SimpleCursorAdapter, 181
- DDMS (Dalvik Debug Monitor Server), xi, 29
- debug mode, detecting, 65
- debug signing key, detecting, 65
- default.properties item, 9
- density-independent pixels (dp), 38, 80–81
- detail\_item.xml layout, creating, 141
- developer tools
  - downloading, 4
  - Draw 9-Patch, 30
  - graphical layout editor, 22–25
  - layoutopt, 30
  - Monkey, 31
- device configuration
  - changes, 62
  - editor, 24
- dialogs
  - confirm, 90–91
  - confirmation, 93
  - creating, 89
  - DialogFragment, 91
  - fragments, 91
  - pre-Android 3.0, 90
- digit.xml layout, using with animation, 240
- dimension units
  - in (inches), 81
  - dip (density-independent pixel), 81
  - dp (density-independent pixel), 81
  - mm (millimeters), 81
  - pt (points), 81
  - px (pixels), 81
  - sp (scaled pixel), 81
- dimensions, placing in res/values folder, 13

- dip (density-independent pixel), 81
- disk I/O, detecting with StrictMode, 64–65
- display, drawing, 258
- dp (density-independent pixels), 38, 80–81
- Draw 9-Patch tool, using, 30, 82
- drawable animations. *See also* animations
  - creating, 232–234
  - ImageView, 233
  - making visible, 232
  - stop() and start(), 233
- drawables
  - Button class, 128
  - StateListDrawable, 128
  - using, 128–129
- drawing display, 258
- DrawingThread class, using with SurfaceView, 284
- drawLines method, using, 265
- DummyTabFactory, using, 177

## E

- Eclipse IDE, x, 4, 9
- edit\_task.xml layout
  - adding button to, 118–119
  - creating, 117–118
- EditText class
  - combining with ToggleButton, 276
  - flags, 116
  - inputType attribute, 116
  - using, 112–114
- email client, opening, 14
- emulator, 6, 8, 26
- event callbacks, 92
- event handling, 92
  - focus events, 95
  - key events, 95
  - long presses, 94
  - main thread, 92
  - onClickListener, 93
  - screen taps, 93
- Example project
  - creating, 4
  - project structure, 9
  - running on phone, 8

## F

- fill parent, 38
- fillAfter attribute, using with animations, 239
- focus events, triggering, 95
- form widgets, availability of, 37. *See also* widgets
- forms
  - Boolean buttons, 119
  - buttons, 118
  - EditText, 112–113
  - fillViewport attribute, 121–123
  - managing settings, 120
  - ScrollView container, 120–121
  - simplifying text entry, 115–118
  - spinners, 119
  - TextView, 112–115
- fragments
  - adding, 160
  - adding to layouts, 154
  - adding to TimeTracker app, 156–159
  - back stack, 161
  - features, 153
  - modifying, 160
  - no-argument constructor, 156
  - onCreate activity callback, 155
  - onDestroy activity callback, 155
  - onPause activity callback, 155
  - onResume activity callback, 155
  - onStart activity callback, 155
  - onStop activity callback, 155
  - removing, 160
  - transactions, 159–160
- FrameLayout container, using with views, 41
- functions, adding via action bar, 171

## G

- gen/ folder, 9
- GestureDetector class
  - onTouchEvent method, 225
  - using, 225–226
- gestures
  - customizing, 228
  - detecting thread violations, 225–226

- GestureDetector class, 225–226
- pinch-to-zoom, 227–228
- ScaleGestureDetector class, 226–228
- getQuantityString method, using in
  - localization, 308
- Google APIs website, 130
- Google Maps, adding permission
  - for, 131
- graphical layout editor, 22–25
  - Canvas, 22
  - Configuration drop-down menu, 22
  - Outline, 23
  - Package Explorer pane, 24
  - Palette, 23
  - tabs, 23
- graphics
  - drawing, 296–300
  - projection matrix, 297–298
  - touch logic, 299–300
  - updateAngle() method, 298–299
- gravity attribute, using with views, 39–40
- GridLayout container
  - adding space, 50
  - margins and padding, 50
  - Spaces, 50–51
  - versus TableLayout, 50
  - using with views, 48–51
- GridView collection view, 206

## H

- Handler class, using with background
  - tasks, 66
- hardware acceleration, adding, 252
- hardware buttons
  - alternative actions, 21
  - Back, 19
  - Home, 19
  - Menu, 20
  - Search, 21
- hardware features, declaring in
  - manifest, 11
- Hello World app
  - Android Device Chooser, 6–7
  - Android emulator, 6, 8

- AVD (Android Virtual Device), 6–7
- Build Target option, 5
- creating, 4–8
- drawable folders, 12
- main.xml file, 16
- package name, 5
- properties, 5
- running, 5
- strings.xml file, 12
- “Hello World!” text, displaying, 218
- Hierarchy Viewer, xi
  - child LinearLayout, 28
  - DDMS Devices pane, 29
  - debugging nested LinearLayouts, 29
  - FrameLayout, 28
  - Layout View, 28
  - left sidebar, 28
  - LinearLayout, 28
  - PhoneWindow, 28
  - Tree Overview, 28
  - Tree View, 28
  - using, 27
- Holo theme, using, 152
- Home button, 19
- home screen
  - app launcher icons, 14
  - grid layout, 196
  - options menu, 20
  - replacement of, 15
  - widgets, 14

## I

- icons
  - creating with Android Asset
    - Studio, 31
  - declaring in manifest, 11
  - quick-launch, 14
- image resources, creating, 31. *See also*
  - resources
- images
  - bitmaps, 127
  - center scale type, 125
  - centerCrop scale type, 125
  - centerInside scale type, 125
  - displaying, 124–129
  - drawables, 128–129

- drawing into views, 129
- fitCenter scale type, 125
- fitEnd scale type, 125
- fitStart scale type, 125
- fitXY scale type, 125
- ImageView and resources, 124–126
- loading, 125
- matrix scale type, 125
- scaleType attribute, 126
- scaling, 125
- ImageView, for drawable
  - animations, 233
- IME (input editor), changing, 115
- in (inches), 81
- inches (in), 81
- <include> tag
  - in detail page, 141–143
  - examples, 140–141
  - layout attribute, 140
  - ls, 140–143
- inflation, explained, 97
- input editor (IME), changing, 115
- Interpolator, ValueAnimator
  - class, 246
- interpolators
  - accelerate, 238
  - bounce, 238
  - decelerate, 238
  - overshoot, 238
  - using with animations, 238
- ISO 639-1 language codes, 304
- ISO 3166-1-alpha-2 region code, 304

## J

- Java perspective, opening in Eclipse, 4

## K

- key events, 95

## L

- labels, declaring in manifest, 11
- language-specific layouts,
  - providing, 306

- layout containers
  - FrameLayout, 41
  - GridLayout, 48–51
  - LinearLayout, 42–45
  - RelativeLayout, 45–47
  - TableLayout, 41–42
  - ViewGroups, 37
- layout resource qualifiers, 77
- layout\_gravity attribute, using, 40
- layout\_margin attribute, using with views, 39
- layout\_weight example, 43
- LayoutInflater class, 97
- layoutopt tool, using, 30
- layouts
  - adding fragments to, 154
  - choosing dimensions for, 38
  - defaults, 52
  - flexibility, 83
  - <include> tag, 140–143
  - inflating, 56
  - life cycle, 154–159
  - margins and padding, 38
  - <merge> tag, 144–146
  - nesting, 144
  - optimizing, 30
  - TextView example, 154–155
  - three buttons, 310–311
  - views in, 17
  - ViewStub class, 146–147
- LayoutTransition class, using, 254
- LinearLayout container
  - orientation setting, 42
  - using with views, 42
- LinearLayout container type, 16–17
  - debugging, 29
  - in Hierarchy Viewer, 28
- list adapters, loading data into, 56
- list navigation mode
  - data binding, 174
  - OnNavigationListener, 174
  - SpinnerAdapter, 174
- list position, saving, 62
- ListActivity, using, 52
- ListAdapter
  - overriding getView method, 54
  - using, 54

- lists
  - binding data to, 54–56
  - displaying, 52–56
  - row layout, 53–54
  - XML layout, 53–54
- ListView
  - attributes, 53
  - collection view, 206–207
  - defining, 53
  - entries attribute, 54
  - using, 52
- Loader helper class, using, 56
- loaders
  - CursorLoader, 185
  - using, 184–185
- localization
  - example, 305
  - formatting, 306–308
  - getQuantityString method, 308
  - ISO 639-1 language codes, 304
  - ISO 3166-1-alpha-2 region code, 304
  - language-specific layouts, 306
  - layouts, 308
  - MCC (mobile country codes), 305
  - MNC (mobile network codes), 305
  - overview, 304–306
  - plurals, 306–308
  - quantity strings, 308
  - resources, 304–306, 308
  - retrieving strings, 308
  - string substitution, 306–308
  - strings, 308
  - testing, 308
  - tips, 308
  - user-visible strings, 308
- long presses, 94
- Looper class, using with background tasks, 66

## M

- main thread
  - avoiding blocking, 66
  - in event handling, 92
  - explained, 61
- manifest entry, using for activities, 57
- map key, registering for, 130
- MapActivity, extending, 132

- maps, adding to applications, 130–135
- MapView, using, 130–132
- margins and padding, 38
- match\_parent, 38
- MCC (mobile country codes), 305
- Menu button, 20, 96
- MenuInflater class, 97
- menus. *See also* action bar
  - assigning icons, 97
  - callbacks, 98–100
  - context, 94, 100–101
  - creating, 96
  - layout inflation, 97
  - layouts, 96–98
  - submenus, 97
- <merge> tag
  - using, 144–146
  - using with ToggleButton, 278
- millimeters (mm), 81
- mm (millimeters), 81
- MNC (mobile network codes), 305
- Monkey tool, features of, 31
- MotionEvent object
  - actions, 219–224
  - using, 219–224

## N

- navigation
  - accessibility, 309–312
  - action bar, 172–174
  - list mode, 174
  - TabWidget interface, 175–178
  - ViewPager behavior, 178
  - ViewPager class, 178–180
- navigation mode, setting, 174
- network I/O, detecting with StrictMode, 64–65
- nextFocus attributes, using for accessibility, 311–312
- notification tray, accessing, 15
- Notification.Builder class, 88
- notifications
  - creating, 86
  - dialogs, 89–91
  - NotificationManager, 87
  - options, 88
  - parameters, 86

- PendingIntent, 86–87
- setLatestEventInfo method, 88
- status bar, 85–88
- toasts, 84–85

## O

- ObjectAnimator class, using with
  - property animations, 247–248
- OnClickInterface, using with screen taps, 93
- OnClickListener interface, 71, 93
- onDraw method, using with custom views, 263–266
- onMeasure method, using with custom views, 260–262
- onSaveInstanceState, using with activities, 62
- onTouchEvent method
  - GestureDetector class, 225
  - implementing, 216–219
  - updating, 221
  - using with SurfaceView, 287
- OpenGL
  - 1.0 standard, 294
  - activity, 295–296
  - drawing graphics, 296–300
  - example, 294–296
  - GLSurfaceView, 294
  - onDrawFrame, 295
  - onSurfaceChanged, 295
  - overview, 294
- options menu functionality, 96
- Outline, described, 23

## P

- Package Explorer pane, 24
- package name, entering, 5
- padding and margins, 38
- padding attribute, using with views, 39
- paging-style interface, using, 180
- Paint object, using with custom views, 263–266
- Palette, described, 23
- permissions, declaring in manifest, 11
- phone dialer, opening, 14

- pinch-to-zoom gesture,
  - implementing, 227–228
- pixel density, variations, 80
- pixels (px), 81
- points (pt), 81
- project structure
  - AndroidManifest.xml item, 9
  - assets/ item, 9
  - default.properties item, 9
  - gen/ folder, 9
  - res/ folder, 9
  - src/ folder, 9, 12–13
- projection matrix, setting, 297–298
- properties, setting, 5
- property animations. *See also* animations
  - AnimatorSet object, 249–250
  - LayoutTransition class, 254
  - ObjectAnimator class, 247–248
  - Property class, 249
  - ValueAnimator class, 246–248
  - ViewPropertyAnimator class, 253
  - in XML, 250–251
- pt (points), 81
- px (pixels), 81

## R

- RelativeLayout container, 45–47
  - child views, 46
  - using, 29
  - XMLAttributes, 47
- RemoteViews class, using, 201–203
- RemoveViewsFactory, using with widgets, 208–209
- RenderScript language, 289–293
  - buildTriangle() method, 293
  - downloading, 290
  - Java API, 291–293
  - RSSurfaceView class, 291
  - sample file, 289–291
  - syntax, 289–290
  - TriangleMeshBuilder, 293
- res/ folder, 9, 12–13
- res/layout folder, contents of, 12
- resource qualifiers, 76–77
  - API version, 77
  - available height, 77

- available width, 77
- precedence, 78–79
- screen orientation, 77
- screen pixel density, 77
- screen size, 77, 79
- SmallestWidth, 77
- resources. *See also* image resources
  - accessing at runtime, 263
  - referencing, 17
- res/values/ folder, contents of, 13
- R.java file, 18–19
- rotating canvas, 272

## S

- scaled pixel (sp), 81
- ScaleGestureDetector class, 226–228
- scaling, using Draw 9-Patch tool for, 30
- screen configuration, determining, 83
- screen orientation, 77
- screen pixel density, 77
- screen sizes, 77, 79
  - Draw 9-Patch tool, 82–83
  - density-independent pixels (dp), 80–81, 83
  - resource qualifiers, 76–80
- screen taps, 93
  - button-click actions, 93
  - confirmation dialog, 93
  - OnClickInterface, 93
  - OnClickListener, 93
- screenshots, taking, 29
- ScrollView container
  - avoiding use of ListView in, 120
  - fillViewport attribute, 121–123
  - using, 120–123
  - wrapping LinearLayout in, 121
- Search button, 21
- settings, managing, 120
- ShareActionProvider class, 171
- SimpleAdapter, using in data binding, 181
- SimpleCursorAdapter, using in data binding, 181
- sliding animations, creating, 244–245.  
*See also* animations
- sp (scaled pixel), 81

- spinners, creating, 119
- src/ folder, 9
- StackView collection view, 206
- startAnimation method, using, 238
- state
  - re-creating for activities, 62
  - saving, 62
- StateListDrawable, using, 128
- status bar notifications, creating, 85–88
- StrictMode
  - declaring, 64–65
  - detecting thread violations, 65
  - disabling, 65
  - enabling, 65
  - explained, 64
- string substitution, using in
  - localization, 306–307
- strings
  - placing in res/values folder, 13
  - referencing, 19
  - retrieving in localization, 308
- strings.xml file, contents of, 12
- styles. *See also* themes
  - adding to TimeTracker app, 150–152
  - attributes, 149
  - defaults, 149
  - defining, 148
  - inheriting, 150
  - <item> elements, 149
  - placing in res/values folder, 13
  - RedText, 148–149
- styles.xml file, creating, 150
- submenus, opening, 97
- sub-views, displaying in linear
  - fashion, 17. *See also* views
- SurfaceView
  - drawing to, 283–288
  - DrawingThread class, 284
  - implementing, 282–283
  - onTouchEvent method, 287
  - rotating triangle, 288
  - versus TextureView class, 288
- system attributes, prefix for, 38

## T

- tab layout, implementing, 175–177
- tabbed interface, creating, 172–173
- table layout, example of, 41
- TableLayout container
  - versus GridLayout, 50
  - using with views, 41–42
- TableRow container, using with
  - views, 41
- TabWidget interface, using, 175–178
- task\_detail.xml, opening, 142
- TaskListFragment class, creating, 156
- tasks. *See also* AsyncTask class
  - back stacks, 62
  - creating, 62
  - grouping activities into, 61–63
  - switching between, 62
- text, zooming, 225–226
- text entry, simplifying, 115–118
- TextureView class versus
  - SurfaceView, 288
- TextView, 17
  - animating, 238–239
  - in clock-flipping animation, 240
  - dragging onto layout, 25
  - using, 112–114
  - using with fragment, 154–155
  - visibility states, 240
- themes, 152. *See also* styles
- threads, running in background, 66
- three-button layout, using for
  - accessibility, 310–311
- time\_row.xml layout file, creating, 53–54
- TimeListAdapter class, creating, 54
- timer, stopping and resetting, 102–107
- timer\_appwidget\_info.xml file,
  - creating, 191
- timer\_widget.xml layout, creating, 194–195
- TimerFragment, creating, 156–158
- TimerWidgetProvider class, creating, 197–200
- TimeTracker app. *See also*
  - background tasks
  - adapter, 183–184
  - AppWidgetProviderInfo XML file, 191–195
  - Boolean buttons, 118
  - button presses, 71–72
  - buttons, 118
  - clearing tasks to, 99–100
  - clock-flipping animation, 240–245
  - confirm dialog, 90–91
  - convenience methods, 67–68
  - creating, 36
  - detail page, 141–143
  - edit\_task.xml layout, 117–118
  - EditText, 113–114
  - fragments, 156–159
  - implementing, 102–107
  - <include> tag, 141–143
  - IntentFilter method, 106
  - layout for list view, 53–54
  - linear layout, 44–45
  - manifest entry for activities, 57
  - notification code, 106
  - OnClickListener interface, 71
  - overriding onClick method, 71
  - Reset button, 72
  - simplifying text entry, 115–118
  - Start/Stop button, 71–72
  - styles, 150–152
  - TextView, 113–114
  - timer layout, 159
  - timer update, 106
  - timerStopped method, 105
  - TimerWidgetProvider class, 197–200
  - tracking time intervals, 66–67
  - updateTime method, 105
  - widget, 191
- toast notification, creating, 84–85
- ToggleButton
  - combining with EditText, 276
  - <merge> tag, 278
  - using Hierarchy Viewer with, 277
- tools
  - downloading, 4
  - Draw 9-Patch, 30
  - graphical layout editor, 22–25
  - layoutopt, 30
  - Monkey, 31



- touch events
  - MotionEvent object, 219–224
  - multi-, 219–223
  - onDraw method, 222
  - onTouchEvent method, 216–219, 221
- touch logic, implementing, 299–300
- touchscreen device, including in
  - manifest, 11
- TranslateAnimation example, 236
- triangle, drawing, 296–297
- TriangleMeshBuilder, using, 293
- TypeEvaluator, ValueAnimator class, 246

## U

- UI thread
  - avoiding blocking, 66
  - in event handling, 92
  - explained, 61
- updateAngle() method,
  - implementing, 298
- UpdateWidgetTime method,
  - implementing, 202

## V

- ValueAnimator class
  - Interpolator, 246
  - TypeEvaluator, 246
  - using with property animations, 246–248
- view animations. *See also* animations
  - alpha option, 235
  - examples, 236–237
  - rotate option, 235
  - scale option, 235
  - translate option, 235
  - TranslateAnimation, 236
- View attributes
  - accessing, 37
  - form of, 38
  - layout\_margin, 39
  - padding, 39
  - specifying, 37–38
- view hierarchy, 41
- View objects
  - Button, 37
  - hierarchy of, 37

- view transparency, changing, 236–237
- ViewGroup
  - layout containers, 37
  - using in custom attributes, 269
- ViewHolder pattern, using, 182–184
- View.Inflate method, 97
- ViewPager class, using, 178–180
- ViewPropertyAnimator class,
  - using, 253
- views. *See also* collection widget;
  - custom views; sub-views;
  - XML view
- Adapter class, 182–183
- adding space around, 38
- arrangement of, 37
- data binding, 181
- dragging and dropping, 22–25
- drawing, 258
- fill parent, 38
- FrameLayout container, 41
- gravity attribute, 39–40
- GridLayout container, 48–51
- height and width, 38
- inflating, 97
- in layouts, 17
- LinearLayout container, 42
- loaders, 184–185
- loading data into, 181–185
- match\_parent, 38
- RelativeLayout container, 45–47
- remote, 201–203
- setting for activities, 18
- TableLayout container, 41–42
- TableRow container, 41
- TextView, 17
- visibility states, 147
- wrap\_content, 38
- ViewStub layout, using, 146–147
- virtual devices, 26
  - creating, 6–7
  - emulated, 26
  - graphics stack, 26
- visibility states
  - applying to drawable
    - animations, 232
  - View.GONE, 147
  - View.INVISIBLE, 147
  - View.VISIBLE, 147

## W

- websites
  - ActionBarSherlock library, 169
  - AppWidgetProviderInfo XML file, 191–195
  - displaying, 130–135
  - enabling plugins, 134
  - Google APIs, 130
  - overriding URLs, 134–135
  - RenderScript API, 290
  - widget template, 190
- WebView class
  - Flash support, 133
  - INTERNET permission, 133
  - using, 133–134
  - WebSettings object, 133
- Widget Preview application, 195
- widget size, calculating, 196
- widget\_background.xml drawable, 194
- widgets, 14. *See also* collection
  - widget; form widgets
  - app template, 190
  - AppWidgetProvider class, 197–200
  - configuration activity, 203–205
  - declaring, 191
  - described, 190
  - layout, 192, 194–195
  - RemoveViews class, 201–203
  - UpdateWidgetTime method, 202
  - wrap\_content, 38

## X

- XML layout, 16–17
  - converting to View objects, 56
  - LinearLayout container type, 16–17
  - setting for activities, 60
- XML layout file, landscape version, 77
- XML view, switching to, 23. *See also*
  - views
- xmlns:android attribute, 16

## Z

- zooming text, 225–226