B.M. Harwani

Features Android Release 4.1.2

Android Programming

UNLEASHED

SAMS

B.M. Harwani

Android™ Programming

UNLEASHED

SAMS

800 East 96th Street, Indianapolis, Indiana 46240 USA

Android™ Programming Unleashed

ISBN-13: 978-0-672-33628-7
ISBN-10: 0-672-33628-6

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

Printed in the United States of America

First Printing: December 2012

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 author 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 or programs accompanying it.

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

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

International Sales
international@pearsoned.com

Editor-in-Chief
Mark Taub

Acquisitions Editor
Laura Lewin

Development Editor
Songlin Qiu

Managing Editor
Kristy Hart

Project Editor
Jovana San
Nicolas-Shirley

Copy Editor
Geneil Breeze

Indexer
Joy Dean Lee

Proofreader
WriteOrWrong
Proofreading Services

Technical Editors
Douglas Jones
Joseph Annuzzi
Roman Irani

Editorial Assistant
Olivia Basegio

Cover Designer
Mark Shirar

Compositor
Gloria Schurick

Contents at a Glance

Introduction 1

Part I Fundamentals of Android Development

1 Introduction to Android........ 9

2 Basic Widgets........ 53

Part II Building Blocks for Android Application Design

3 Laying Out Controls in Containers 101

4 Utilizing Resources and Media........ 147

5 Using Selection Widgets and Debugging........ 209

6 Displaying and Fetching Information Using Dialogs and Fragments........ 259

Part III Building Menus and Storing Data

7 Creating Interactive Menus and ActionBars........ 323

8 Using Databases........ 385

Part IV Advanced Android Programming: Internet, Entertainment, and Services

9 Implementing Drawing and Animation 421

10 Displaying Web Pages and Maps 473

11 Communicating with SMS and Emails........ 517

12 Creating and Using Content Providers........ 559

13 Creating and Consuming Services 591

14 Publishing Android Applications........ 633

Index........ 643

Table of Contents

Introduction **1**

Key Topics That This Book Covers....1
Key Benefits That This Book Provides 2
How This Book Is Organized 2
Code Examples for This Book 5

I: Fundamentals of Android Development

1 Introduction to Android **9**

The Android 4.1 Jelly Bean SDK....9
Understanding the Android Software Stack 11
Installing the Android SDK....12
- Adding Platforms and Other Components....16
- Installing Eclipse....20
- Installing the Android Development Tools (ADT) Plug-in....22
- Making the ADT Plug-in Functional....24

Creating Android Virtual Devices 24
Creating the First Android Project....27
- Laying Out the Application 31
- Defining Action Code Through Java....34
- Running the Application....36

Using the `TextView` Control....39
- Assigning the Text Directly in the Layout File....39
- Assigning Text Through the Activity File....40
- Applying Dimensions to Controls....42
- Aligning Content with the Gravity Attribute....43
- Commonly Used Attributes....43

Using the Android Emulator 47
- Limitations of the Android Emulator....47

The Android Debug Bridge (ADB) 48
Launching Android Applications on a Handset....50
Summary....51

2 Basic Widgets **53**

Understanding the Role of Android Application Components 53
Understanding the Utility of Android API....55
Overview of the Android Project Files....57
Understanding Activities 59
- Understanding the Android Activity Life Cycle 60

Role of the Android Manifest File 61
- Using the Manifest Editor 64

Creating the User Interface....64
Commonly Used Layouts and Controls....66

Event Handling....67
Creating an Anonymous Inner Class....68
Activity Implementing the OnClickListener Interface....71
Declaring the Event Handler in the XML Control Definition....72
Displaying Messages Through Toast....75
Creating and Starting an Activity....76
Describing Operations Through Intent....77
Method Used to Start an Activity....77
Creating Your Own Layout File....78
Creating a New Activity....79
Registering the New Activity....80
Starting the Activity....81
Using the EditText Control....82
Attributes Used to Configure the EditText Control....82
Adding an Event Listener to the EditText Control....84
Choosing Options with CheckBox....87
Choosing Mutually Exclusive Items Using RadioButtons....91
Summary....98

II: Building Blocks for Android Application Design

3 Laying Out Controls in Containers 101
Introduction to Layouts....101
LinearLayout....102
Applying the orientation Attribute....102
Applying the height and width Attributes....103
Applying the padding Attribute....103
Applying the weight Attribute....106
Applying the Gravity Attribute....108
Using the android:layout_gravity Attribute....109
RelativeLayout....111
Layout Control Attributes....113
AbsoluteLayout....121
Using ImageView....124
FrameLayout....125
TableLayout....129
Operations Applicable to TableLayout....129
GridLayout Layout....133
Specifying Row and Column Position....134
Spanning Rows and Columns....134
Inserting Spaces in the GridLayout....134
Adapting to Screen Orientation....138
Anchoring Controls....138
Defining Layout for Each Mode....140
Summary....145

4 Utilizing Resources and Media 147

Resources 147
Types of Resources....148
Creating Values Resources....150
Dimension Resources 153
Color Resources 156
Styles and Themes 159
Applying Themes....162
Arrays 165
Using Drawable Resources....170
Switching States with Toggle Buttons....174
Creating an Image Switcher Application....179
Scrolling Through ScrollView 183
Use of the android:fillViewport Attribute....185
Playing Audio....186
Adding Audio to the Application....187
Playing Video....195
Loading Video onto an SD Card 195
Displaying Progress with ProgressBar....199
Using Assets 204
Summary....207

5 Using Selection Widgets and Debugging 209

Using ListView....209
Creating a ListView with an Activity Base Class 211
Creating ListView by Extending ListActivity....217
Using the Spinner Control 220
Populating a Spinner Through Resources....220
Populating a Spinner Through ArrayAdapter....223
AutoCompleteTextView....225
Using the GridView Control....227
GridView Attributes....228
Displaying Images in GridView....231
Creating an Image Gallery Using the ViewPager Control 235
Using the Debugging Tool: Dalvik Debug Monitor Service (DDMS)....239
Debugging Applications 245
Placing Breakpoints in an Application 245
Using the Debug Perspective....247
Debug Pane 248
Expressions Pane....249
Breakpoints Pane....250
Variables Pane 254
Adding Logging Support to Android Applications 255
Summary....256

6 Displaying and Fetching Information Using Dialogs and Fragments 259

What Are Dialogs? 259
AlertDialog....261
Methods of the AlertDialog.Builder Subclass....261
Getting Input via the Dialog Box 264

DatePickerDialog 267
TimePickerDialog 271
Selecting the Date and Time in One Application 275
Fragments 282
The Structure of a Fragment 282
The Life Cycle of a Fragment....282
Creating Fragments with Java Code 294
FragmentManager....294
Communicating Between Fragments 296
Navigating to Previous Fragments....296
Retrieving Content Passed Through Bundle 297
Saving and Restoring the State of Fragments 297
Creating Special Fragments 301
Creating a ListFragment 301
Using a DialogFragment 305
Using PreferenceFragment 311
Summary....319

III: Building Menus and Storing Data

7 Creating Interactive Menus and ActionBars 323
Menus and Their Types 323
Creating Menus Through XML 324
Creating an Options Menu 325
Adding Submenus 332
Creating a Context Menu 336
Creating Menus Through Coding....345
Defining Options Menus 345
Creating Submenus....346
Trying It Out....349
Applying a Context Menu to a ListView....354
Using the ActionBar 358
Enabling the ActionBar 360
Using an Application's Icon for Navigation 361
Displaying Action Items 362
Replacing a Menu with the ActionBar 370
Creating a Tabbed ActionBar....377
Creating a Drop-Down List ActionBar 380
Summary....384

8 Using Databases 385
Using the SQLiteOpenHelper Class....385
Building an SQLite Project 386
Fetching the Desired Rows from Tables....391
Using Cursors 392
Accessing Databases with the ADB....394
Accessing the Database Through Menus 398
Creating a Data Entry Form....401
Displaying Table Rows Via ListView....410
Summary....417

IV: Advanced Android Programming: Internet, Entertainment, and Services

9 Implementing Drawing and Animation 421

Drawing on the Screen 421
- Using `Canvas` and `Paint` 421
- Using Gradients 436

Animations 445
- Understanding Frame-by-Frame Animation 446
- Understanding Tweening Animation 453

Applying Interpolators 471
Summary 472

10 Displaying Web Pages and Maps 473

Displaying Web Pages 473
- Enabling JavaScript 477
- Handling Page Navigation 477
- Adding Permission for Internet Access 478

Using the `WebViewClient` Class 480
Using Google Maps 483
- Obtaining a Google Maps API Key 483
- Installing the Google API 484
- AVDs for Map-Based Applications 485
- Creating a Google Maps-Based Application 486
- Using Location-Based Services 490
- Supplying Latitude and Longitude Values Through DDMS 494
- Sending GPS Locations Manually 495
- Passing Locations in GPX/KML Format 496
- Displaying Map Locations 496
- Printing the GPS Location Address 502
- Displaying Map Markers 507
- Using `ItemizedOverlay` 511

Summary 516

11 Communicating with SMS and Emails 517

Understanding Broadcast Receivers 517
- Broadcasting an `Intent` 518
- Receiving the Broadcast `Intent` 519

Using the Notification System 523
- Notification via the Status Bar 523

Sending SMS Messages with Java Code 531
- Getting Permission to Send SMS Messages 534
- Writing Java Code 534

Receiving SMS Messages 541
Sending Email 546

Working with the Telephony Manager........553
Making the Outgoing Call........553
Listening for Phone State Changes........554
Summary........558

12 Creating and Using Content Providers 559
What Is a Content Provider........559
Understanding the Android Content URI........560
Using Content Providers........561
Creating a Custom Content Provider........566
Defining a Content Provider........566
Defining a Database........568
Defining the Content URI........569
Defining MIME Types........570
Implementing the `getType`, `query`, `insert`, `update`, and `delete` Methods........571
Registering Content Providers........587
Summary........589

13 Creating and Consuming Services 591
Moving Tasks to Background Threads........591
Using the `Handler` Class........592
Using the `AsyncTask` Class........594
Accessing Data from the Internet........598
Consuming SOAP Services........602
Creating a Service........607
Interacting with the Service........611
Creating a Bound Service........614
Setting Up Alarms........619
Setting Repeating Alarms........620
Setting Up the Time for the Alarm........620
Using Sensors........626
Summary........631

14 Publishing Android Applications 633
Setting Versioning Information of an Application........633
Generating a Certificate, Digitally Signing the Android Applications, and Generating the APK........636
Signing Applications Using the Export Android Application Wizard........637
Distributing Applications with Google Play........638
Getting Started with Google Play........639
Localizing Android Applications........641
Monetizing Our Applications........642
Summary........642

Index 643

This page intentionally left blank

About the Author

B.M. Harwani is founder and owner of Microchip Computer Education (MCE), based in Ajmer, India, that provides computer education in all programming and web developing platforms. He graduated with a BE in computer engineering from the University of Pune, and also has a C Level (master's diploma in computer technology) from DOEACC, Government of India. Being involved in the teaching field for more than 18 years, he has developed the art of explaining even the most complicated topics in a straightforward and easily understandable fashion. To know more, visit his blog http://bmharwani.com/blog.

Dedication

Dedicated to my mother, Mrs. Nita Harwani, Ray Tomlinson, and Dr. V. A. Shiva Ayyadurai.

My mother is next to God for me. Whatever I am today is just because of the moral values taught by her.

I admire and appreciate Ray Tomlinson and Dr. V. A. Shiva Ayyadurai's invention—Internet-based email.
They have revolutionized the mode of communication. In fact, their achievement has changed the life of millions of people around the world, including me.

Acknowledgments

I owe a debt of gratitude to Laura Lewin, Acquisitions Editor, for her initial acceptance and giving me an opportunity to create this work. I am highly grateful to the whole team at Pearson Technology Group for their constant cooperation and contribution to create this book.

My gratitude to Songlin Qiu, who as a Development Editor, offered a significant amount of feedback that helped to improve the chapters. She played a vital role in improving the structure and quality of information.

I must thank Douglas Jones, Joseph Annuzzi, and Romin Irani, the Technical Reviewers, for their excellent, detailed reviewing of the work and the many helpful comments and suggestions they made.

Special thanks to Geneil Breeze, Copy Editor, for first-class structural and language editing. I appreciate her efforts in enhancing the contents of the book and giving it a polished look.

I also thank Gloria Schurick, Compositor, for doing excellent formatting and making the book dramatically better.

Big and ongoing thanks to Jovana Shirley, Project Editor, for doing a great job and for the sincere efforts of the whole team to get the book published on time.

A great big thank you to the editorial and production staff and the entire team at Pearson Technology Group who worked tirelessly to produce this book. Really, I enjoyed working with each of you.

I am also thankful to my family, my small world: Anushka (my wife) and my two little darlings, Chirag and Naman, for inspiring me to work harder.

I should not forget to thank my dear students who have been good teachers for me as they make me understand the basic problems they face in a subject and enable me to directly hit at those topics. The endless interesting queries of my students help me to write books with a practical approach.

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: errata@informit.com

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

Reader Services

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

Introduction

Android is Google's open source and free Java-based platform for mobile development. It enables developers to build real-world mobile applications using the Android SDK and publish them on Google Play.

The huge demand for developing Android applications inspired me to write this book. Like any good book, it begins by explaining the usage of basic UI controls one at a time, configuring them by applying different attributes, and writing applications to understand how they respond to user actions. Gradually, once the reader is acquainted with the basic UI controls, the book explains how to use the advanced controls, resources, dialogs, and different types of menus.

The book addresses intermediate to advanced users and teaches different components provided by the Android SDK through examples. The book will be beneficial for developers and instructors too who want to learn or teach Android programming. For practical implementation the book also explains using the back-end databases for storing and fetching information. In short it is a useful reference book for anyone who wants to understand all key aspects of Android programming and to apply them practically into developing Android applications.

Key Topics That This Book Covers

This book is comprehensive and covers each topic in detail. Key topics covered are

- Understanding basic controls and event handling.
- Using resources, media, audio, and video.
- Creating of different types of menus with XML as well as through Java code.
- Accessing databases in Android applications.
- Using Internet, Google Maps, and Location-Based Services.
- Different types of layouts and selection widgets.
- Sending and receiving SMS messages and emails.
- Everything required for developing applications—for example, UI controls, containers, databases, menus—and accessing the Internet is available in one place.
- The book is completely up to date with the latest Jelly Bean.

Key Benefits That This Book Provides

By the time you finish the book, you will be able to

- Use and configure UI controls to develop Android applications
- Understand the technique of organizing controls in different layouts
- Use different resources in developing feature-rich Android applications
- Use different dialogs for getting data from the user
- Store, fetch, and update database records, and to access databases through menus
- Display web pages and Google Maps
- Send and receive SMS messages and emails
- Use the Telephony Manager for making phone calls
- Create your own custom service and also learn to consume SOAP Services
- Draw graphics, apply animation, and use interpolators
- Create, use, and register Content Providers
- Execute events automatically through Alarm Manager
- Use device sensors
- Publish Android applications

How This Book Is Organized

This book is structured in four parts:

- Part I: "Fundamentals of Android Development"

  In Chapter 1, "Introduction to Android," you learn to install the Android SDK Starter Package, add platforms and other components, and install Eclipse and the Android Developer Tools (ADT) plug-in. You learn to make the ADT plug-in functional and create Android Virtual Devices to run and deploy Android applications. You also learn to create and run your first Android project, and you learn to set the layout of the application and the usage of the `TextView` control in an Android application.

  Chapter 2, "Basic Widgets," focuses on the basic widgets used in an Android application. You learn about folders and files that are automatically created by the ADT plug-in, activities, the Android Activity life cycle, usage of the Android Manifest file, commonly used layouts and controls, and how event handling is performed. You learn how to create an anonymous inner class, implement the `OnClickListener` interface, and declare the event handler in the XML definition of the control. The chapter shows how to create a new Activity, register the new Activity, and start the Activity, and how to use three controls—EditText, CheckBox, and RadioButton—to develop Android applications.

- Part II: “Building Blocks for Android Application Design”

In Chapter 3, “Laying Out Controls in Containers,” you learn about containers—the different types of layouts used to organize and arrange the controls of an application. You learn to use LinearLayout, RelativeLayout, AbsoluteLayout, FrameLayout, and TableLayout, and you learn to adapt to the screen orientation. In addition, you learn the usage of different attributes that help in laying out controls in different layouts. The chapter shows you how to apply different attributes in the layouts such as the Orientation attribute, Height and Width attribute, Padding attribute, Weight attribute, and Gravity attribute.

Chapter 4, “Utilizing Resources and Media,” discusses the different types of resources and the procedures to apply them in Android applications. You learn to apply Dimension resources, Color resources, styles, and themes. You also learn to use String and Integer arrays. To display images in an Android application, you learn to use Drawable resources and create an Image Switcher application using the ToggleButton control. Also, you learn to implement scrolling through ScrollView and to play audio and video. Finally, the chapter explains using ProgressBar and assets.

Chapter 5, “Using Selection Widgets and Debugging,” focuses on selection widgets. You learn to use the ListView, Spinner, AutoComplete, and GridView controls in Android applications. You learn how to use display options in selection widgets through string arrays and the ArrayAdapter, and you also see how to extend ListActivity and use styling for the standard ListAdapters. You learn to create an Image Gallery using Gallery Control and the procedure to use the debugging tool, Dalvik Debug Monitor Service (DDMS). The chapter also explains the procedure involved in debugging applications, placing breakpoints in an application, and using Debug perspective. And you learn to adding logging support to Android applications.

In Chapter 6, “Displaying and Fetching Information Using Dialogs and Fragments,” you learn to use different dialogs in Android applications. You learn to use the AlertDialog to display important messages to the user, as well as to receive input from the user. You also learn to display and select dates and times with the DatePicker and TimePicker dialog boxes. The chapter explains fragments, their life cycles, and the procedure for creating them through XML and with Java code. You also learn about specialized fragments: ListFragment, DialogFragment, and PreferenceFragment.

- Part III: “Building Menus and Storing Data”

  In Chapter 7, “Creating Interactive Menus and ActionBars,” you learn about different types of menus. You learn to create options menus, expanded menus, submenus, and context menus with XML as well as Java code. You also learn to use check boxes/radio buttons in menus, handle menu selections, add shortcut keys, and assign icons to menu items. You learn to use the ActionBar, display action items, and create a tabbed ActionBar and a drop-down list ActionBar.

  In Chapter 8, “Using Databases,” you learn to use databases in Android applications. In the chapter you use the SQLite `SQLiteOpenHelper` to fetch desired rows from a table, and you learn to use cursors. You also learn to access databases through ADB and menus, and you learn to create data entry forms and display table rows through `ListView`.

- Part IV: “Advanced Android Programming: Internet, Entertainment, and Services”

  Chapter 9, “Implementing Drawing and Animation,” focuses on understanding animation. You learn to use `Canvas` and `Paint`, measure screen coordinates, and apply frame-by-frame animation. You also learn about tweening animation and the use of interpolators.

  In Chapter 10, “Displaying Web Pages and Maps,” you learn to display web pages through `WebView` controls, handle page navigation, and add permissions for Internet access. You see how to use the `WebViewClient`, use Google Maps, get Google Keys, and install the Google API. You learn to create AVDs for map-based applications, use location-based services, supply latitude and longitude values through DDMS, add zooming, and display map markers.

  In Chapter 11, “Communicating with SMS and Emails,” you learn about broadcast receivers. You see how to broadcast and receive the broadcasted `intent`. You also see how the Notification system is used, created, configured, and displayed in the status bar. You learn the procedure for sending and receiving SMS messages programmatically. Finally, you learn how to send email and use the Telephony Manager to make phone calls.

  In Chapter 12, “Creating and Using Content Providers,” you learn how to define, create, use, and register Content Providers. You also learn to define a database, Content URI, and MIME types. Also you learn to implement the `getType`, `query`, `insert`, `update`, and `delete` methods. Finally, the chapter explains how to use loaders.

  In Chapter 13, “Creating and Consuming Services,” you learn to move processes to the background threads using the `Handler` and `AsyncTask` classes. You learn to download and display images from the Internet. The chapter also explains how to create your own Bind Service and the procedure to consume SOAP Services. You also learn to use Alarm and Sensor Managers.

In Chapter 14, “Publishing Android Applications,” you learn how to publish Android applications. You learn about versioning and digitally signing your applications, deploying APK files, and publishing your applications to the Google Play Store.

Code Examples for This Book

All the Android projects discussed in this book are available to download from the www.informit.com/title/ 9780672336287. Download the code bundle provided in the site and unzip it. Follow these steps to use the provided code:

1. Launch Eclipse.
2. Select the `File`, `Import` option. From the `Import` dialog that opens, select the `Existing Projects into Workspace` option and click the `Next` button.
3. In the next dialog, click the `Browse` button to locate and select the folder where you unzipped the code bundle.
4. After you select the code bundle, all the Android projects enclosed in it appear in the `Projects` box. By default all the projects are checked. Uncheck projects that you don’t want to import and click `Finish`. That’s it. The projects are imported into Eclipse and are ready to run.

This page intentionally left blank

CHAPTER 3

Laying Out Controls in Containers

IN THIS CHAPTER

- Introduction to Layouts
- LinearLayout
- Applying the `Orientation` Attribute
- Applying `Height` and `Width` Attributes
- Applying the `Padding` Attribute
- Applying the `Weight` attribute
- Applying the `Gravity` Attribute
- Using the `android:layout_gravity` Attribute
- RelativeLayout
- Relative Layout Control Attributes
- AbsoluteLayout
- FrameLayout
- TableLayout
- TableLayout Operations
- GridLayout
- Screen Orientation Adaptations

A container is a view used to contain other views. Android offers a collection of view classes that act as containers for views. These container classes are called layouts, and as the name suggests, they decide the organization, size, and position of their children views.

Let's start the chapter with an introduction to different layouts used in Android applications.

Introduction to Layouts

Layouts are basically containers for other items known as `Views`, which are displayed on the screen. Layouts help manage and arrange views as well. Layouts are defined in the form of XML files that cannot be changed by our code during runtime.

Table 3.1 shows the layout managers provided by the Android SDK.

TABLE 3.1 Android Layout Managers

Layout Manager	Description
LinearLayout	Organizes its children either horizontally or vertically
RelativeLayout	Organizes its children relative to one another or to the parent
AbsoluteLayout	Each child control is given a specific location within the bounds of the container

Layout Manager	Description
FrameLayout	Displays a single view; that is, the next view replaces the previous view and hence is used to dynamically change the children in the layout
TableLayout	Organizes its children in tabular form
GridLayout	Organizes its children in grid format

The containers or layouts listed in Table 3.1 are also known as `ViewGroups` as one or more `Views` are grouped and arranged in a desired manner through them. Besides the `ViewGroups` shown here Android supports one more `ViewGroup` known as ScrollView, which is discussed in Chapter 4, "Utilizing Resources and Media."

LinearLayout

The LinearLayout is the most basic layout, and it arranges its elements sequentially, either horizontally or vertically. To arrange controls within a linear layout, the following attributes are used:

- **`android:orientation`**—Used for arranging the controls in the container in horizontal or vertical order
- **`android:layout_width`**—Used for defining the width of a control
- **`android:layout_height`**—Used for defining the height of a control
- **`android:padding`**—Used for increasing the whitespace between the boundaries of the control and its actual content
- **`android:layout_weight`**—Used for shrinking or expanding the size of the control to consume the extra space relative to the other controls in the container
- **`android:gravity`**—Used for aligning content within a control
- **`android:layout_gravity`**—Used for aligning the control within the container

Applying the `orientation` Attribute

The `orientation` attribute is used to arrange its children either in horizontal or vertical order. The valid values for this attribute are `horizontal` and `vertical`. If the value of the `android:orientation` attribute is set to `vertical`, the children in the linear layout are arranged in a column layout, one below the other. Similarly, if the value of the `android:orientation` attribute is set to `horizontal`, the controls in the linear layout are arranged in a row format, side by side. The orientation can be modified at runtime through the `setOrientation()` method. That is, by supplying the values `HORIZONTAL` or `VERTICAL` to the `setOrientation()` method, we can arrange the children of the LinearLayout in row or column format, respectively.

Applying the height and width Attributes

The default height and width of a control are decided on the basis of the text or content that is displayed through it. To specify a certain height and width to the control, we use the android:layout_width and android:layout_height attributes. We can specify the values for the height and width attributes in the following three ways:

- By supplying specific dimension values for the control in terms of px (pixels), dip/dp (device independent pixels), sp (scaled pixels), pts (points), in (inches), and mm (millimeters). For example, the android:layout_width="20px" attribute sets the width of the control to 20 pixels.
- By providing the value as wrap_content. When assigned to the control's height or width, this attribute resizes the control to expand to fit its contents. For example, when this value is applied to the width of the TextView, it expands so that its complete text is visible.
- By providing the value as match_parent. When assigned to the control's height or width, this attribute forces the size of the control to expand to fill up all the available space of the enclosing container.

NOTE

For layout elements, the value wrap_content resizes the layout to fit the controls added as its children. The value match_parent makes the layout expand to take up all the space in the parent layout.

Applying the padding Attribute

The padding attribute is used to increase the whitespace between the boundaries of the control and its actual content. Through the android:padding attribute, we can set the same amount of padding or spacing on all four sides of the control. Similarly, by using the android:paddingLeft, android:paddingRight, android:paddingTop, and android:paddingBottom attributes, we can specify the individual spacing on the left, right, top, and bottom of the control, respectively.

The following example sets the spacing on all four sides of the control to 5 pixels:

```
android:padding="5dip"
```

Similarly, the following example sets the spacing on the left side of the control to 5 pixels:

```
android:paddingLeft="5dip"
```

NOTE

To set the padding at runtime, we can call the setPadding() method.

Let's see how the controls are laid out in the LinearLayout layout using an example. Create a new Android Project called `LinearLayoutApp`. The original default content of the layout file `activity_linear_layout_app.xml` appears as shown in Listing 3.1.

LISTING 3.1 Default Code in the Layout File `activity_linear_layout_app.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".LinearLayoutAppActivity" />
</RelativeLayout>
```

Let's apply the LinearLayout and add three `Button` controls to the layout. Modify the `activity_linear_layout_app.xml` to appear as shown in Listing 3.2.

LISTING 3.2 The `activity_linear_layout_app.xml` File on Adding Three `Button` Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The orientation of LinearLayout is set to vertical, declaring that we want to arrange its child elements vertically, one below the other. The height and width of the layout are set to expand to fill up all the available space of the enclosing container, that is, the device screen. Three Button controls are added to the layout, which appear one below the other. The IDs and text assigned to the three Button controls are Apple, Mango, and Banana, respectively. The height of the three controls is set to wrap_content, which is enough to accommodate the text. Finally, the width of the three controls is set to match_parent, so that the width of the three controls expands to fill up the available space of the LinearLayout container. We see the output shown in Figure 3.1.

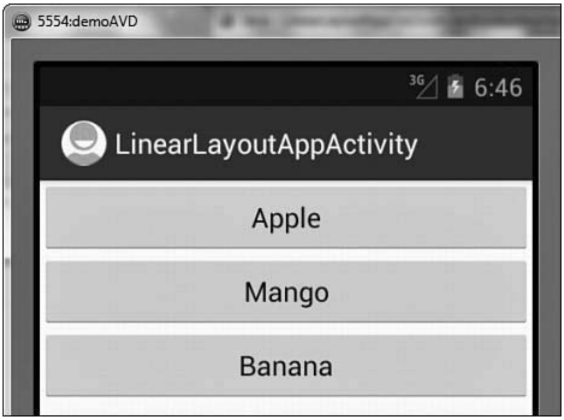


FIGURE 3.1 Three Button controls arranged vertically in LinearLayout

To see the controls appear horizontally, set the orientation attribute of the LinearLayout to horizontal. We also need to set the layout_width attribute of the three controls to wrap_content; otherwise, we will be able to see only the first Button control, the one with the Apple ID. If the layout_width attribute of any control is set to match_parent, it takes up all the available space of the container, hiding the rest of the controls behind it. By setting the values of the layout_width attributes to wrap_content, we make sure that the width of the control expands just to fit its content and does not take up all the available space. Let's modify the activity_linear_layout_app.xml to appear as shown in Listing 3.3.

LISTING 3.3 The activity_linear_layout_app.xml File on Setting Horizontal Orientation to the Button Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The controls are arranged horizontally, as shown in Figure 3.2.

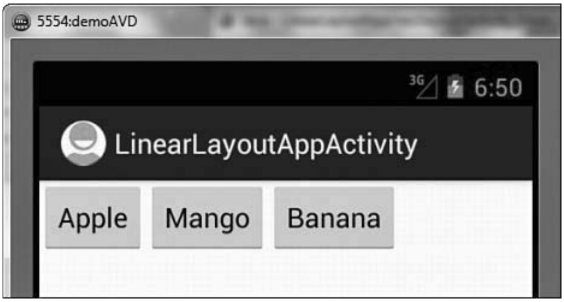


FIGURE 3.2 Three `Button` controls arranged horizontally in LinearLayout

Applying the `weight` Attribute

The `weight` attribute affects the size of the control. That is, we use `weight` to assign the capability to expand or shrink and consume extra space relative to the other controls in the container. The values of the `weight` attribute range from `0.0` to `1.0`, where `1.0` is the highest value. Let's suppose a container has two controls and one of them is assigned the `weight` of `1`. In that case, the control assigned the `weight` of `1` consumes all the empty space in the container, whereas the other control remains at its current size. If we assign a `weight` of `0.0` to both the controls, nothing happens and the controls maintain their original size. If both the attributes are assigned the same value above `0.0`, both the controls consume the extra space equally. Hence, `weight` lets us apply a size expansion ratio to the controls. To make the middle `Button` control, `Mango`, take up all the available space of the container, let's assign a `weight` attribute to the three controls. Modify the `activity_linear_layout_app.xml` file to appear as shown in Listing 3.4.

LISTING 3.4 The `activity_linear_layout_app.xml` File on Applying the `weight` Attribute to the `Button` Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1.0" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />
</LinearLayout>
```

By setting the `layout_weight` attributes of `Apple`, `Mango`, and `Banana` to `0.0`, `1.0`, and `0.0`, respectively, we allow the `Mango` button control to take up all the available space of the container, as shown in Figure 3.3 (left). If we set the value of `layout_weight` of the `Banana` button control to `1.0` and that of `Mango` back to `0.0`, then all the available space of the container is consumed by the `Banana` button control, as shown in Figure 3.3 (middle). Similarly if we set the `layout_weight` of all controls to `1.0`, the entire container space will be equally consumed by the three controls, as shown in Figure 3.3 (right).

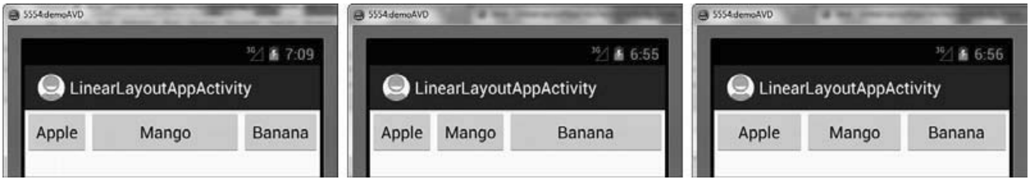


FIGURE 3.3 (left) The `weight` attribute of the Mango `Button` control set to 1.0, (middle) the `weight` attribute of the Banana `Button` control set to 1.0, and (right) all three `Button` controls set to the same `weight` attribute

Similarly if we set the weight of `Apple`, `Mango`, and `Banana` to `0.0`, `1.0`, and `0.5`, respectively, we get the output shown in Figure 3.4.

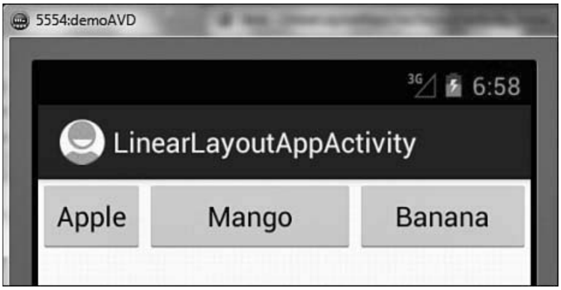


FIGURE 3.4 The `weight` attribute of the Apple, Mango, and Banana `Button` controls set to 0.0, 1.0, and 0.5

We can see that the text of the three controls is center-aligned. To align the content of a control, we use the `Gravity` attribute.

Applying the `Gravity` Attribute

The `Gravity` attribute is for aligning the content within a control. For example, to align the text of a control to the center, we set the value of its `android:gravity` attribute to `center`. The valid options for `android:gravity` include `left`, `center`, `right`, `top`, `bottom`, `center_horizontal`, `center_vertical`, `fill_horizontal`, and `fill_vertical`. The task performed by few of the said options is as follows:

- **`center_vertical`**—Places the object in the vertical center of its container, without changing its size
- **`fill_vertical`**—Grows the vertical size of the object, if needed, so it completely fills its container
- **`center_horizontal`**—Places the object in the horizontal center of its container, without changing its size
- **`fill_horizontal`**—Grows the horizontal size of the object, if needed, so it completely fills its container
- **`center`**—Places the object in the center of its container in both the vertical and horizontal axis, without changing its size

We can make the text of a control appear at the center by using the `android:gravity` attribute, as shown in this example:

```
android:gravity="center"
```

We can also combine two or more values of any attribute using the | operator. The following example centrally aligns the text horizontally and vertically within a control:

```
android:gravity="center_horizontal|center_vertical"
```

Figure 3.5 shows the `android:gravity` attribute set to left and right for the `Button` controls `Mango` and `Banana`.

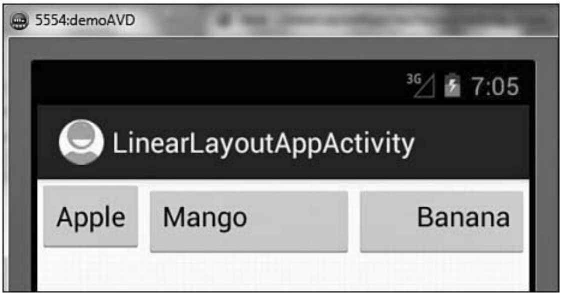


FIGURE 3.5 The text in the Mango and Banana Button controls aligned to the left and right, respectively, through the android:gravity attribute

Besides the android:gravity attribute, Android provides one more similar attribute, android:layout_gravity. Let's explore the difference between the two.

Using the android:layout_gravity Attribute

Where android:gravity is a setting used by the View, the android:layout_gravity is used by the container. That is, this attribute is used to align the control within the container. For example, to align the text within a Button control, we use the android:gravity attribute; to align the Button control itself in the LinearLayout (the container), we use the android:layout_gravity attribute. Let's add the android:layout_gravity attribute to align the Button controls themselves. To see the impact of using the android:layout_gravity attribute to align the Button controls in the LinearLayout, let's first arrange them vertically. So, let's modify activity_linear_layout_app.xml to make the Button controls appear vertically, one below the other as shown in Listing 3.5.

LISTING 3.5 The activity_linear_layout_app.xml File on Arranging the Button Controls Vertically

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The preceding code arranges the Button controls vertically, as shown in Figure 3.6 (left). To align the Button controls Mango and Banana to the center and to the right of the LinearLayout container, add the following statements to the respective tags in the activity_linear_layout_app.xml layout file:

```
android:layout_gravity="center"
```

and

```
android:layout_gravity="right"
```

The two Button controls, Mango and Banana, are aligned at the center and to the right in the container, as shown in Figure 3.6 (middle).

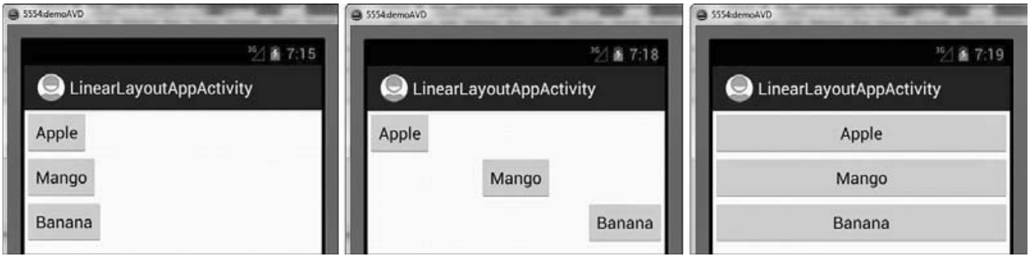


FIGURE 3.6 (left) The three Button controls vertically aligned with the width attribute set to wrap_content, (middle) the Mango and Banana Button controls aligned to the center and right of container, and (right) the width of the three Button controls expanded to take up all the available space

At the moment, the layout_width attribute of the three controls is set to wrap_content. The width of the three controls is just enough to accommodate their content. If we now set the value of the android:layout_width attribute for all three controls to match_parent, we find that all three Button controls expand in width to take up all the available space of the container, as shown in Figure 3.6 (right). Now we can apply the android:gravity attribute to align the text within the controls. Let's add the following three attributes to the Button controls Apple, Mango, and Banana:

```
android:gravity="left"
android:gravity="center"
```

and

```
android:gravity="right"
```

These lines of code align the content of the three Button controls to the left, to the center, and to the right within the control, as shown in Figure 3.7 (left). Because the three Button controls are arranged vertically in the layout (the orientation of the LinearLayout is set to vertical), the application of the weight attribute makes the controls expand vertically instead of horizontally as we saw earlier. To see the effect, let's add the following statement to the tags of all three `Button` controls:

```
android:layout_weight="0.0"
```

As expected, there will be no change in the height of any control, as the `weight` value assigned is `0.0`. Setting an equal value above `0.0` for all three controls results in equal division of empty space among them. For example, assigning the `android:layout_weight="1.0"` to all three controls results in expanding their height, as shown in Figure 3.7 (middle).

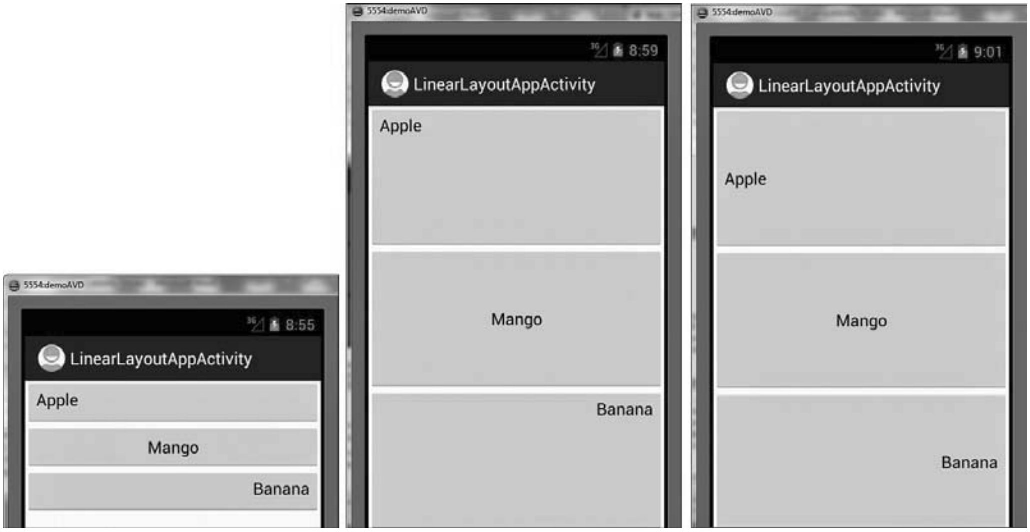


FIGURE 3.7 (left) The three `Button` controls with their text aligned to the left, center, and right, (middle) the vertical available space of the container apportioned equally among the three `Button` controls, and (right) the text of the three `Button` controls vertically aligned to the center

In the middle image of Figure 3.7, we see that the text in the `Apple` and `Banana` controls is not at the vertical center, so let's modify their `android:gravity` value, as shown here:

`android:gravity="center_vertical"` for the `Apple` control

`android:gravity="center_vertical|right"` for the `Banana` control

The `center_vertical` value aligns the content vertically to the center of the control, and the `right` value aligns the content to the right of the control. We can combine the values of the attribute using the `|` operator. After applying the values as shown in the preceding two code lines, we get the output shown in Figure 3.7 (right).

RelativeLayout

In RelativeLayout, each child element is laid out in relation to other child elements; that is, the location of a child element is specified in terms of the desired distance from the existing children. To understand the concept of relative layout practically, let's create a new Android project called RelativeLayoutApp. Modify its layout file activity_relative_layout_app.xml to appear as shown in Listing 3.6.

LISTING 3.6 The activity_relative_layout_app.xml File on Arranging the Button Controls in the RelativeLayout Container

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dip"
        android:layout_marginLeft="20dip" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="28dip"
        android:layout_toRightOf="@id/Apple"
        android:layout_marginLeft="15dip"
        android:layout_marginRight="10dip"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="200dip"
        android:layout_height="50dip"
        android:layout_marginTop="15dip"
        android:layout_below="@id/Apple"
        android:layout_alignParentLeft="true" />
    <Button
        android:id="@+id/Grapes"
        android:text="Grapes"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:minWidth="100dp"
        android:layout_alignParentRight="true"
        android:layout_below="@id/Banana" />
    <Button
        android:id="@+id/Kiwi"
        android:text="Kiwi"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_below="@id/Banana"
        android:paddingTop="15dip"
        android:paddingLeft="25dip"
        android:paddingRight="25dip" />
</RelativeLayout>
```

Before we understand how the controls in the previous code block are placed, let's have a quick look at different attributes used to set the positions of the layout controls.

Layout Control Attributes

The attributes used to set the location of the control relative to a container are

- **`android:layout_alignParentTop`**—The top of the control is set to align with the top of the container.
- **`android:layout_alignParentBottom`**—The bottom of the control is set to align with the bottom of the container.
- **`android:layout_alignParentLeft`**—The left side of the control is set to align with the left side of the container.
- **`android:layout_alignParentRight`**—The right side of the control is set to align with the right side of the container.
- **`android:layout_centerHorizontal`**—The control is placed horizontally at the center of the container.
- **`android:layout_centerVertical`**—The control is placed vertically at the center of the container.
- **`android:layout_centerInParent`**—The control is placed horizontally and vertically at the center of the container.

The attributes to control the position of a control in relation to other controls are

- **`android:layout_above`**—The control is placed above the referenced control.
- **`android:layout_below`**—The control is placed below the referenced control.
- **`android:layout_toLeftOf`**—The control is placed to the left of the referenced control.
- **`android:layout_toRightOf`**—The control is placed to the right of the referenced control.

The attributes that control the alignment of a control in relation to other controls are

- `android:layout_alignTop`— The top of the control is set to align with the top of the referenced control.

- **`android:layout_alignBottom`**—The bottom of the control is set to align with the bottom of the referenced control.
- **`android:layout_alignLeft`**—The left side of the control is set to align with the left side of the referenced control.
- **`android:layout_alignRight`**—The right side of the control is set to align with the right side of the referenced control.
- **`android:layout_alignBaseline`**—The baseline of the two controls will be aligned.

For spacing, Android defines two attributes: `android:layout_margin` and `android:padding`. The `android:layout_margin` attribute defines spacing for the container, while `android:padding` defines the spacing for the view. Let's begin with padding.

- **`android:padding`**—Defines the spacing of the content on all four sides of the control. To define padding for each side individually, use `android:paddingLeft`, `android:paddingRight`, `android:paddingTop`, and `android:paddingBottom`.
- **`android:paddingTop`**—Defines the spacing between the content and the top of the control.
- **`android:paddingBottom`**—Defines the spacing between the content and the bottom of the control.
- **`android:paddingLeft`**—Defines the spacing between the content and the left side of the control.
- **`android:paddingRight`**—Defines the spacing between the content and the right side of the control.

Here are the attributes that define the spacing between the control and the container:

- **`android:layout_margin`**—Defines the spacing of the control in relation to the controls or the container on all four sides. To define spacing for each side individually, we use the `android:layout_marginLeft`, `android:layout_marginRight`, `android:layout_marginTop`, and `android:layout_marginBottom` options.
- **`android:layout_marginTop`**—Defines the spacing between the top of the control and the related control or container.
- **`android:layout_marginBottom`**—Defines the spacing between the bottom of the control and the related control or container.
- **`android:layout_marginRight`**—Defines the spacing between the right side of the control and the related control or container.
- **`android:layout_marginLeft`**—Defines the spacing between the left side of the control and the related control or container.

The layout file activity_relative_layout_app.xml arranges the controls as follows:

> The Apple button control is set to appear at a distance of 15dip from the top and 20dip from the left side of the RelativeLayout container. The width of the Mango button control is set to consume the available horizontal space. The text Mango appears at a distance of 28dip from all sides of the control. The Mango control is set to appear to the right of the Apple control. The control is set to appear at a distance of 15dip from the control on the left and 10dip from the right side of the relative layout container. Also, the top of the Button control is set to align with the top of the container.
>
> The Banana button control is assigned the width and height of 200dip and 50dip, respectively. The control is set to appear 15dip below the Apple control. The left side of the control is set to align with the left side of the container.
>
> The Grapes button control is set to appear below the Banana button control, and its width is set to expand just enough to accommodate its content. The height of the control is set to take up all available vertical space. The text Grapes is automatically aligned vertically; that is, it appears at the center of the vertical height when the height attribute is set to match_parent. The minimum width of the control is set to 100dip. The right side of the control is set to align with the right side of the container.
>
> The Kiwi Button control is set to appear below the Banana control. Its width is set to 100dip, and the height is set to just accommodate its content. The text Kiwi is set to appear at the distance of 15dip, 25dip, and 25dip from the top, left, and right boundary of the control.

We don't need to make any changes to the RelativeLayoutAppActivity.java file. Its original content is as shown in Listing 3.7.

LISTING 3.7 The Default Code in the Activity File RelativeLayoutAppActivity.java

```java
package com.androidunleashed.relativelayoutapp;

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

public class RelativeLayoutDemoActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_relative_layout_app);
    }
}
```

When the application is run, we see the output shown in Figure 3.8.

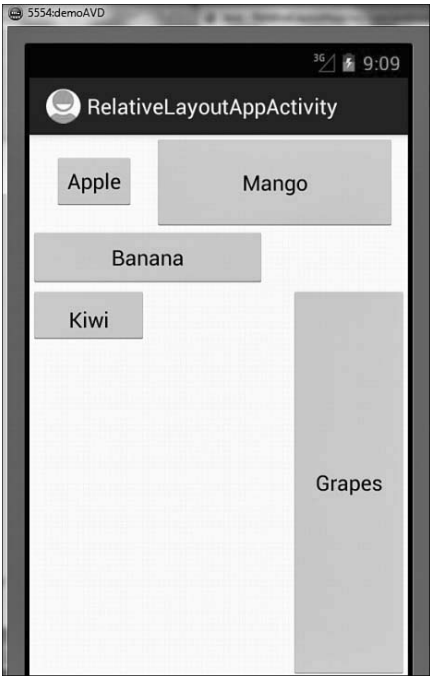


FIGURE 3.8 The five `Button` controls' layout relative to each other

We can make the text `Grapes` appear centrally at the top row by adding the following line:

```
android:gravity="center_horizontal"
```

So, its tag appears as follows:

```
<Button
    android:id="@+id/Grapes"
    android:text="Grapes"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:minWidth="100dp"
    android:layout_alignParentRight="true"
    android:layout_below="@id/Banana"
    android:gravity="center_horizontal" />
```

The output is modified to appear as shown in Figure 3.9.

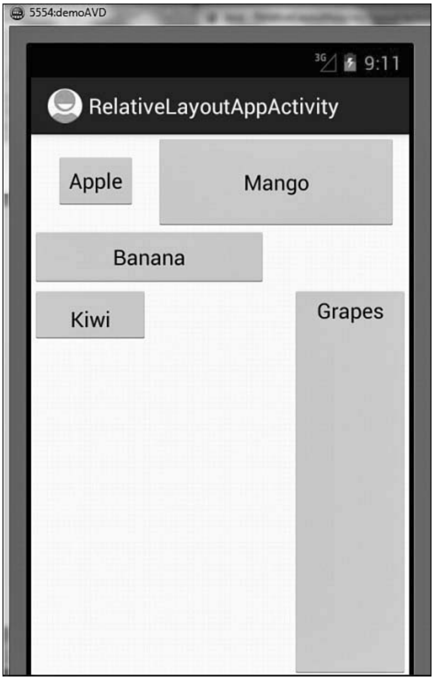


FIGURE 3.9 The Grapes `Button` control aligned horizontally at the center

Let's explore the concept of laying out controls in the RelativeLayout container by writing an application. The application that we are going to create is a simple `Login Form` application that asks the user to enter a `User ID` and `Password`. The `TextView`, `EditText`, and `Button` controls in the application are laid out in a RelativeLayout container (see Figure 3.10—left). If either the `User ID` or `Password` is left blank, the message `The User ID or password is left blank. Please Try Again` is displayed. If the correct `User ID` and `Password`, in this case, `guest`, are entered, then a welcome message is displayed. Otherwise, the message `The User ID or password is incorrect. Please Try Again` is displayed.

So, let's create the application. Launch the Eclipse IDE and create a new Android application called `LoginForm`. Arrange four `TextView` controls, two `EditText` controls, and a `Button` control in RelativeLayout, as shown in the layout file `activity_login_form.xml` displayed in Listing 3.8.

LISTING 3.8 The `activity_login_form.xml` on Laying Out the `TextView`, `EditText`, and `Button` Controls in the RelativeLayout Container

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/sign_msg"
        android:text = "Sign In"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:typeface="serif"
        android:textSize="25dip"
        android:textStyle="bold"
        android:padding="10dip"
        android:layout_centerHorizontal="true"/>
    <TextView
        android:id="@+id/user_msg"
        android:text = "User ID:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:layout_below="@+id/sign_msg" />
    <EditText
        android:id="@+id/user_ID"
        android:layout_height="wrap_content"
        android:layout_width="250dip"
        android:layout_below="@+id/sign_msg"
        android:layout_toRightOf="@+id/user_msg"
        android:singleLine="true" />
    <TextView
        android:id="@+id/password_msg"
        android:text = "Password:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/user_msg"
        android:layout_margin="10dip"
        android:paddingTop="10dip"/>
    <EditText
        android:id="@+id/password"
        android:layout_height="wrap_content"
        android:layout_width="250dp"
        android:singleLine="true"
        android:layout_below="@+id/user_ID"
        android:layout_toRightOf="@+id/password_msg"
        android:password="true" />
    <Button
        android:id="@+id/login_button"
        android:text="Sign In"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dip"
        android:layout_below="@+id/password_msg"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/response"
        android:layout_below="@+id/login_button"/>
</RelativeLayout>
```

The controls in the application are arranged in the RelativeLayout, as explained here:

- Through the `TextView` control `sign_msg`, the text `Sign In` is displayed horizontally centered at the top. It is displayed in bold serif font, `25 dip` in size. The text is padded with a space of `10dip` on all four sides of its container.
- Another `TextView` control, `user_msg`, displays the text `User ID` below the `TextView sign_msg`. The `TextView` is placed `10dip` from all four sides.
- An `EditText` control `user_ID` is displayed below `sign_msg` and to the right of `user_msg`. The width assigned to the `TextView` control is `250 dip` and is set to `single-line` mode, so if the user types beyond the given width, the text scrolls to accommodate extra text but does not run over to the second line.
- A `TextView password_msg` control displaying the text `Password:` is displayed below the `TextView user_msg`. The `TextView` control is placed at a spacing of `10dip` from all four sides, and the text `Password:` is displayed at `10dip` from the control's top boundary.
- An `EditText` control `password` is displayed below the `EditText user_ID` and to the right of the `TextView password_msg`. The `width` assigned to the `TextView` control is `250` dip and is set to `single-line` mode. In addition, the typed characters are converted into dots for security.
- A `Button` control `login_button` with the caption `Sign In` is displayed below the `TextView password_msg`. The button is horizontally centered and is set to appear at `10dip` distance from the `EditText` control `password`.
- A `TextView` control `response` is placed below the `Button login_button`. It is used to display messages to the user when the `Sign In` button is pressed after entering `User ID` and `Password`.

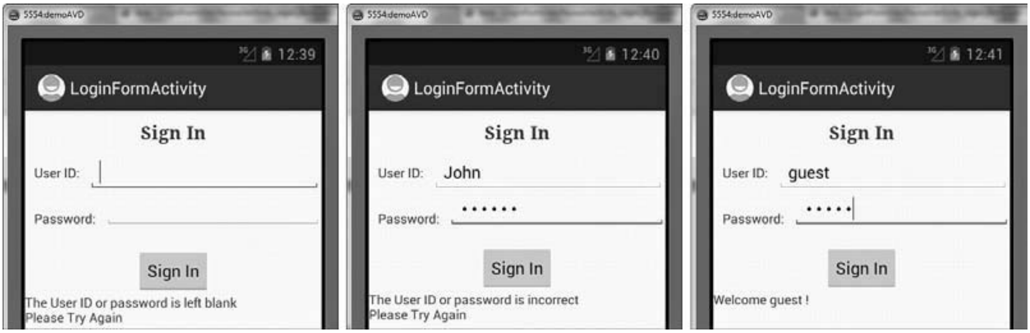
To authenticate the user, we need to access the `User ID` and `Password` that is entered and match these values against the valid `User ID` and `Password`. In addition, we want to validate the `EditText` controls to confirm that none of them is blank. We also want to welcome the user if he or she is authorized. To do all this, we write the code in the activity file `LoginFormActivity.java` as shown in Listing 3.9.

LISTING 3.9 Code Written in the Java Activity File `LoginFormActivity.java`

```
package com.androidunleashed.loginform;

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

public class LoginFormActivity extends Activity implements OnClickListener  {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login_form);
        Button b = (Button)this.findViewById(R.id.login_button);
        b.setOnClickListener(this);
    }

    public void onClick(View v) {
        EditText userid = (EditText) findViewById(R.id.user_ID);
        EditText password = (EditText) findViewById(R.id.password);
        TextView resp = (TextView)this.findViewById(R.id.response);
        String usr = userid.getText().toString();
        String pswd = password.getText().toString();
        if(usr.trim().length() == 0 || pswd.trim().length() == 0){
            String str = "The User ID or password is left blank \nPlease Try Again";
            resp.setText(str);
        }
        else{
            if(usr.equals("guest") && pswd.equals("guest")) resp.setText("Welcome " +
            usr+ " ! ");
           else resp.setText("The User ID or password is incorrect \nPlease Try Again");
        }
    }
}
```

The `Button` control is accessed from the layout file and is mapped to the `Button` object `b`. This activity implements the `OnClickListener` interface. Hence, the class implements the callback method `onClick()`, which is invoked when a click event occurs on the `Button` control.

In the `onClick()` method, the `user_ID` and `password` `EditText` controls are accessed from the layout file and mapped to the `EditText` objects `userid` and `password`. Also, the `TextView` control `response` is accessed from the layout file and is mapped to the `TextView` object `resp`. The `User ID` and `password` entered by the user in the two `EditText` controls are accessed through the objects `userid` and `password` and assigned to the two Strings `usr` and `pswd`, respectively. The data in the `usr` and `pswd` strings is checked for authentication. If the user has left any of the `EditText` controls blank, the message `The User ID or password is left blank. Please Try Again` is displayed, as shown in Figure 3.10 (left). If the `User ID` and `password` are correct, then a welcome message is displayed (see Figure 3.10—right). Otherwise, the message `The User ID or password is incorrect. Please Try Again` is displayed, as shown in Figure 3.10 (middle).



The User ID or password is left blank
Please Try Again

John
••••••

The User ID or password is incorrect
Please Try Again

guest
•••••

Welcome guest !

FIGURE 3.10 (left) The Login Form displays an error if fields are left blank, (middle) the Password Incorrect message displays if the user ID or password is incorrect, and (right) the Welcome message displays when the correct user ID and password are entered.

AbsoluteLayout

Each child in an AbsoluteLayout is given a specific location within the bounds of the container. Such fixed locations make AbsoluteLayout incompatible with devices of different screen size and resolution. The controls in AbsoluteLayout are laid out by specifying their exact *X* and *Y* positions. The coordinate 0,0 is the origin and is located at the top-left corner of the screen.

Let's write an application to see how controls are positioned in AbsoluteLayout. Create a new Android Project called `AbsoluteLayoutApp`. Modify its layout file, `activity_absolute_layout_app.xml`, as shown in Listing 3.10.

LISTING 3.10 The Layout File `activity_absolute_layout_app.xml` on Arranging Controls in the AbsoluteLayout Container

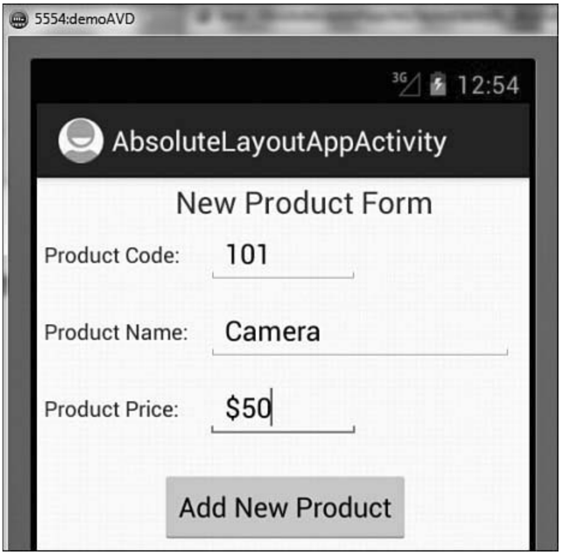
```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Product Form"
        android:textSize="20sp"
        android.textStyle="bold"
        android:layout_x="90dip"
        android:layout_y="2dip"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Code:"
        android:layout_x="5dip"
        android:layout_y="40dip" />
    <EditText
        android:id="@+id/product_code"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="100dip"
        android:layout_x="110dip"
        android:layout_y="30dip" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Name:"
        android:layout_x="5dip"
        android:layout_y="90dip"/>
    <EditText
        android:id="@+id/product_name"
        android:layout_width="200dip"
        android:layout_height="wrap_content"
        android:minWidth="200dip"
        android:layout_x="110dip"
        android:layout_y="80dip"
        android:scrollHorizontally="true" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Price:"
        android:layout_x="5dip"
        android:layout_y="140dip" />
    <EditText
        android:id="@+id/product_price"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="100dip"
        android:layout_x="110dip"
        android:layout_y="130dip" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/click_btn"
        android:text="Add New Product"
        android:layout_x="80dip"
        android:layout_y="190dip" />
</AbsoluteLayout>
```

The controls in `activity_absolute_layout_app.xml` are as follows:

- The `New Product Form TextView` is set to appear `90dip` from the left and `2dip` from the top side of the container. The size of the text is set to `20sp`, and its style is set to `bold`.
- The `Product Code TextView` is set to appear `5dip` from the left and `40dip` from the top side of the container.
- The `product_code EditText` control is set to appear `110dip` from the left and `30dip` from the top side of the container. The minimum width of the control is set to `100dp`.
- The `ProductName TextView` control is set to appear `5dip` from the left and `90dip` from the top side of the container.
- The `product_name EditText` control is set to appear `110dip` from the left and `80dip` from the top side of the container. The minimum width of the control is set to `200dip`, and its text is set to scroll horizontally when the user types beyond its width.
- The `Product Price TextView` is set to appear `5dip` from the left and `140dip` from the top side of the container.
- The `product_price EditText` control is set to appear `110dip` from the left and `130dip` from the top side of the container. The minimum width of the control is set to `100dip`.
- The `click_btn Button`, `Add New Product`, is set to appear `80dip` from the left and `190dip` from the top side of the container.

If we don’t specify the x, y coordinates of a control in AbsoluteLayout, it is placed in the origin point, that is, at location 0,0. If the value of the x and y coordinates is too large, the control does not appear on the screen. The values of the x and y coordinates are specified in any units, such as `sp`, `in`, `mm`, and `pt`.

After specifying the locations of controls in the layout file `activity_absolute_layout_app.xml`, we can run the application. There is no need to make any changes in the file `AbsoluteLayoutAppActivity.java`. When the application is run, we get the output shown in Figure 3.11.



New Product Form

Product Code: 101

Product Name: Camera

Product Price: $50

FIGURE 3.11 Different controls laid out in AbsoluteLayout

The AbsoluteLayout class is not used often, as it is not compatible with Android phones of different screen sizes and resolutions.

The next layout we are going to discuss is FrameLayout. Because we will learn to display images in FrameLayout, let's first take a look at the `ImageView` control that is often used to display images in Android applications.

Using `ImageView`

An `ImageView` control is used to display images in Android applications. An image can be displayed by assigning it to the `ImageView` control and including the `android:src` attribute in the XML definition of the control. Images can also be dynamically assigned to the `ImageView` control through Java code.

A sample ImageView tag when used in the layout file is shown here:

```
<ImageView
    android:id="@+id/first_image"
    android:src = "@drawable/bintupic"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitXY"
    android:adjustViewBounds="true"
    android:maxHeight="100dip"
    android:maxWidth="250dip"
    android:minHeight="100dip"
    android:minWidth="250dip"
    android:resizeMode="horizontal|vertical" />
```

Almost all attributes that we see in this XML definition should be familiar, with the exception of the following ones:

- **android:src**—Used to assign the image from drawable resources. We discuss drawable resources in detail in Chapter 4. For now, assume that the image in the res/drawable folder is set to display through the ImageView control via this attribute.

  Example:

  ```
  android:src = "@drawable/bintupic"
  ```

  You do not need to specify the image file extension. JPG and GIF files are supported, but the preferred image format is PNG.

- **android:scaleType**—Used to scale an image to fit its container. The valid values for this attribute include fitXY, center, centerInside, and fitCenter. The value fitXY independently scales the image around the X and Y axes without maintaining the aspect ratio to match the size of container. The value center centers the image in the container without scaling it. The value centerInside scales the image uniformly, maintaining the aspect ratio so that the width and height of the image fit the size of its container. The value fitCenter scales the image while maintaining the aspect ratio, so that one of its X or Y axes fits the container.

- **android:adjustViewBounds**—If set to true, the attribute adjusts the bounds of the ImageView control to maintain the aspect ratio of the image displayed through it.

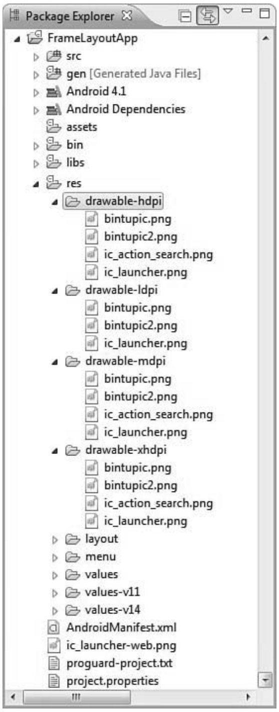
- **android:resizeMode**—The resizeMode attribute is used to make a control resizable so we can resize it horizontally, vertically, or around both axes. We need to click and hold the control to display its resize handles. The resize handles can be dragged in the desired direction to resize the control. The available values for the resizeMode attribute include horizontal, vertical, and none. The horizontal value resizes the control around the horizontal axis, the vertical value resizes around the vertical axis, the both value resizes around both the horizontal and vertical axes, and the value none prevents resizing.

FrameLayout

FrameLayout is used to display a single View. The View added to a FrameLayout is placed at the top-left edge of the layout. Any other View added to the FrameLayout overlaps the previous View; that is, each View stacks on top of the previous one. Let's create an application to see how controls can be laid out using FrameLayout.

In the application we are going to create, we will place two ImageView controls in the FrameLayout container. As expected, only one ImageView will be visible, as one ImageView will overlap the other ImageView, assuming both ImageView controls are of the same size. We will also display a button on the ImageView, which, when selected, displays the hidden ImageView underneath.

Let's start with the application. Create a new Android project called `FrameLayoutApp`. To display images in Android applications, the image is first copied into the `res/drawable` folder and from there, it is referred to in the layout and other XML files. We look at the procedure for displaying images, as well as the concept of drawable resources, in detail in Chapter 4. For the time being, it is enough to know that to enable the image(s) to be referred to in the layout files placed in the `res/drawable` folder, the image needs to exist in the `res/drawable` folder. There are four types of drawable folders: `drawable-xhdpi`, `drawable-hdpi`, `/res/drawable-mdpi`, and `/res/drawable-ldpi`. We have to place images of different resolutions and sizes in these folders. The graphics with the resolutions `320 dpi`, `240dpi`, `160 dpi`, and `120dpi` (`96 x 96 px`, `72 x 72 px`, `48 x 48 px`, and `36 x 36 px`), are stored in the `res/drawable-xhdpi`, `res/drawable-hdpi`, `res/drawable-mdpi`, and `res/drawable-ldpi` folders, respectively. The application picks up the appropriate graphic from the correct folder. So, if we copy two images called `bintupic.png` and `bintupic2.png` of the preceding size and resolution and paste them into the four `res/drawable` folders, the `Package Explorer` resembles Figure 3.12.



```
Package Explorer
FrameLayoutApp
  src
  gen [Generated Java Files]
  Android 4.1
  Android Dependencies
  assets
  bin
  libs
  res
    drawable-hdpi
      bintupic.png
      bintupic2.png
      ic_action_search.png
      ic_launcher.png
    drawable-ldpi
      bintupic.png
      bintupic2.png
      ic_launcher.png
    drawable-mdpi
      bintupic.png
      bintupic2.png
      ic_action_search.png
      ic_launcher.png
    drawable-xhdpi
      bintupic.png
      bintupic2.png
      ic_action_search.png
      ic_launcher.png
    layout
    menu
    values
    values-v11
    values-v14
  AndroidManifest.xml
  ic_launcher-web.png
  proguard-project.txt
  project.properties
```

FIGURE 3.12 The `Package Explorer` window showing the two images, `bintupic.png` and `bintupic2.png`, dropped into the `res/drawable` folders

To display two `ImageView`s and a `TextView` in the application, let's write the code in the layout file `activity_frame_layout_app.xml` as shown in Listing 3.11.

LISTING 3.11 The Layout File `activity_frame_layout_app.xml` on Arranging the `ImageView` and `TextView` Controls in the FrameLayout Container

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:id="@+id/first_image"
        android:src = "@drawable/bintupic"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY" />
    <ImageView
        android:id="@+id/second_image"
        android:src = "@drawable/bintupic2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click the image to switch"
        android:layout_gravity="center_horizontal|bottom"
        android:padding="5dip"
        android:textColor="#ffffff"
        android:textStyle="bold"
        android:background="#333333"
        android:layout_marginBottom="10dip" />
</FrameLayout>
```

The `first_image` and `second_image` `ImageView` controls are set to display the images `bintupic.png` and `bintupic2.png`, respectively. To make the two images stretch to cover the entire screen, the `scaleType` attribute in the `ImageView` tag is set to `fitXY`. A `TextView`, `Click the image to switch`, is set to display at the horizontally centered position and at a distance of `10dip` from the bottom of the container. The spacing between the text and the boundary of the `TextView` control is set to `5dip`. The background of the text is set to a dark color, the foreground color is set to white, and its style is set to bold. When a user selects the current image on the screen, the image should switch to show the hidden image. For this to occur, we need to write code in the activity file as shown in Listing 3.12.

LISTING 3.12 Code Written in the Java Activity File `FrameLayoutAppActivity.java`

```
package com.androidunleashed.framelayoutapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.view.View.OnClickListener;
import android.view.View;

public class FrameLayoutAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_frame_layout_app);
        final ImageView first_image = (ImageView)this.findViewById(R.id.first_image);
        final ImageView second_image = (ImageView)this.findViewById(R.id.second_image);
        first_image.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                second_image.setVisibility(View.VISIBLE);
                view.setVisibility(View.GONE);
            }
        });
        second_image.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                first_image.setVisibility(View.VISIBLE);
                view.setVisibility(View.GONE);
            }
        });
    }
}
```

The two `first_image` and `second_image` `ImageView` controls are located through the `findViewById` method of the Activity class and assigned to the two `ImageView` objects, `first_image` and `second_image`, respectively. We register the click event by calling the `setOnClickListener()` method with an `OnClickListener`. An anonymous listener is created on the fly to handle click events for the `ImageView`. When the `ImageView` is clicked, the `onClick()` method of the listener is called. In the `onClick()` method, we switch the images; that is, we make the current `ImageView` invisible and the hidden `ImageView` visible. When the application runs, we see the output shown in Figure 3.13 (left). The application shows an image, and the other image is hidden behind it because in FrameLayout one View overlaps the other. When the user clicks the image, the images are switched, as shown in Figure 3.13 (right).

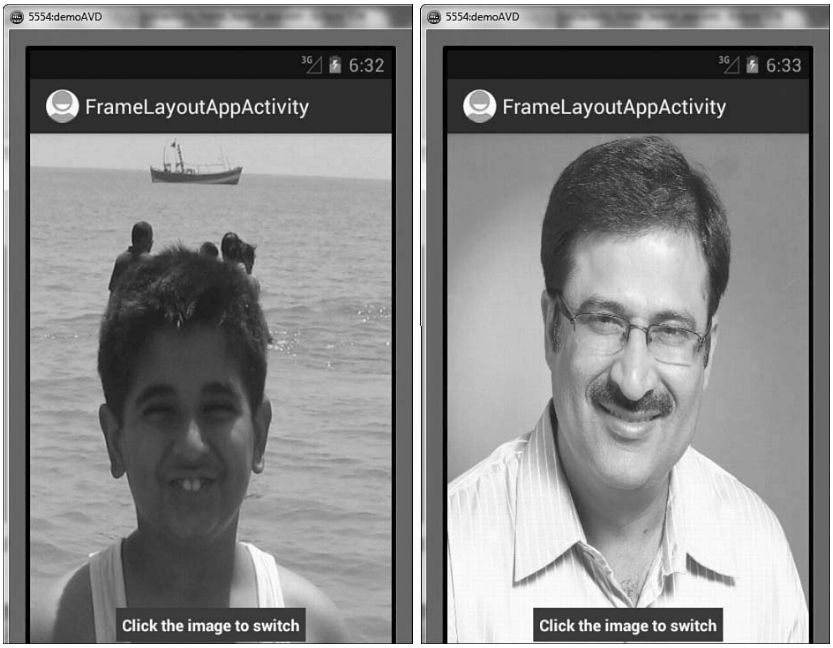


FIGURE 3.13 (left) An image and a `TextView` laid out in FrameLayout, and (right) the images switch when clicked

TableLayout

The TableLayout is used for arranging the enclosed controls into rows and columns. Each new row in the TableLayout is defined through a `TableRow` object. A row can have zero or more controls, where each control is called a `cell`. The number of columns in a TableLayout is determined by the maximum number of cells in any row. The width of a column is equal to the widest cell in that column. All elements are aligned in a column; that is, the width of all the controls increases if the width of any control in the column is increased.

NOTE

We can nest another TableLayout within a table cell, as well.

Operations Applicable to TableLayout

We can perform several operations on TableLayout columns, including stretching, shrinking, collapsing, and spanning columns.

Stretching Columns

The default width of a column is set equal to the width of the widest column, but we can stretch the column(s) to take up available free space using the `android:stretchColumns` attribute in the TableLayout. The value assigned to this attribute can be a single column number or a comma-delimited list of column numbers. The specified columns are stretched to take up any available space on the row.

Examples:

- **`android:stretchColumns="1"`**—The second column (because the column numbers are zero-based) is stretched to take up any available space in the row.
- **`android:stretchColumns="0,1"`**—Both the first and second columns are stretched to take up the available space in the row.
- **`android:stretchColumns="*"`**—All columns are stretched to take up the available space.

Shrinking Columns

We can shrink or reduce the width of the column(s) using the `android:shrinkColumns` attribute in the TableLayout. We can specify either a single column or a comma-delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

NOTE

By default, the controls are not word-wrapped.

Examples:

- **`android:shrinkColumns="0"`**—The first column's width shrinks or reduces by word-wrapping its content.
- **`android:shrinkColumns="*"`**—The content of all columns is word-wrapped to shrink their widths.

Collapsing Columns

We can make the column(s) collapse or become invisible through the `android:collapseColumns` attribute in the TableLayout. We can specify one or more comma-delimited columns for this attribute. These columns are part of the table information but are invisible. We can also make column(s) visible and invisible through coding by passing the `Boolean` values `false` and `true`, respectively, to the `setColumnCollapsed()` method in the TableLayout. For example:

- **`android:collapseColumns="0"`**—The first column appears collapsed; that is, it is part of the table but is invisible. It can be made visible through coding by using the `setColumnCollapsed()` method.

Spanning Columns

We can make a column span or take up the space of one or more columns by using the `android:layout_span` attribute. The value assigned to this attribute must be `>=1`. For example, the following value makes the control take or span up to two columns:

```
android:layout_span="2"
```

Let's try arranging controls in a TableLayout with an example. Create a new Android project called `TableLayoutApp`. Make its layout file `activity_table_layout_app.xml` appear as shown in Listing 3.13.

LISTING 3.13 The Layout File `activity_table_layout_app.xml` on Arranging Controls in a TableLayout Container

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">
    <TableRow android:padding="5dip">
        <TextView
            android:layout_height="wrap_content"
            android:text="New Product Form"
            android:typeface="serif"
            android:layout_span="2"
            android:gravity="center_horizontal"
            android:textSize="20dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_height="wrap_content"
            android:text="Product Code:"
            android:layout_column="0"/>
        <EditText
            android:id="@+id/prod_code"
            android:layout_height="wrap_content"
            android:layout_column="1"/>
    </TableRow>
    <TableRow>
        <TextView
            android:layout_height="wrap_content"
            android:text="Product Name:"
            android:layout_column="0"/>
        <EditText
            android:id="@+id/prod_name"
            android:layout_height="wrap_content"
            android:scrollHorizontally="true" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_height="wrap_content"
            android:text="Product Price:" />
        <EditText
            android:id="@+id/prod_price"
            android:layout_height="wrap_content" />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/add_button"
            android:text="Add Product"
            android:layout_height="wrap_content" />
        <Button
            android:id="@+id/cancel_button"
            android:text="Cancel"
            android:layout_height="wrap_content" />
    </TableRow>
</TableLayout>
```

We cannot specify the `layout_width` attribute for the controls enclosed within the TableLayout, as their width will be always set to `match_parent` by default. We can specify the `layout_height` attribute for the enclosed controls (the default value is `wrap_content`). The `layout_height` attribute of the `TableRow` is always `wrap_content`.

Cells are added to a row in increasing column order. Column numbers are zero-based. If we don't specify a column number for any cell, it is considered to be the next available column. If we skip a column number, it is considered an empty cell in that row. We can make a cell span columns. Besides `TableRow`, we can use any `View` subclass as a direct child of `TableLayout`. The `View` is displayed as a single row that spans all the table columns.

NOTE

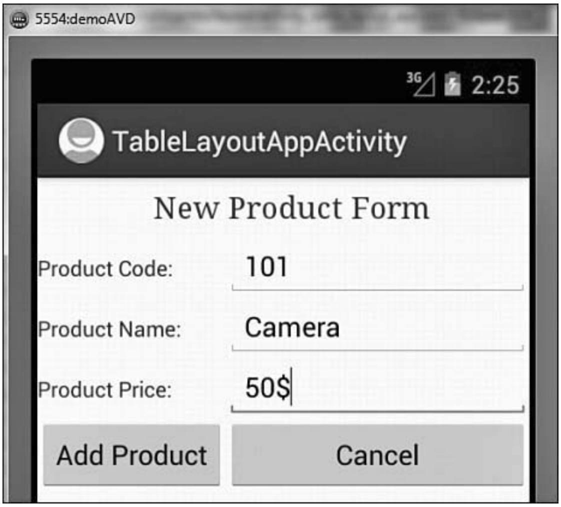
TableLayout does not display border lines for rows, columns, or cells.

In Listing 3.13, we specify that the second column of each row should be stretched to take up any available space in the row. The row contents are

- The first row of the table has a single control, `New Product Form TextView`. The `TextView` is set to span two columns and is set to appear at the center of the horizontal space. The `font` of the text displayed through `TextView` is set to `serif`, `20dip` in size.

- In the second row, a TextView and an EditText control are displayed. The TextView control with text Product Code is set to appear at the column 0 location (the first column), and the EditText control is set to appear at column 1 (the second column).
- In the third row, again two controls, TextView and EditText, are displayed. The TextView control with the text Product Name is set to appear in column 0. If the user types text beyond the width of the EditText control, the content scrolls horizontally.
- In the fourth row, the TextView control with the text Product Price is displayed in the first column, and the EditText control is displayed in the second column.
- In the fifth row, a Button control with the caption Add Product is displayed in the first column, and a Button control with the caption Cancel is displayed in the second column.

When the application is run, the controls are laid out in rows and columns, as shown in Figure 3.14.



5554:demoAVD

2:25

TableLayoutAppActivity

New Product Form

Product Code: 101

Product Name: Camera

Product Price: 50$

FIGURE 3.14 Different controls arranged in TableLayout

GridLayout Layout

GridLayout lays out views in a two-dimensional grid pattern, that is, in a series of rows and columns. The intersection of row and column is known as a grid cell, and it is the place where child views are placed. It is easier to use GridLayout when compared to TableLayout. Without specifying intermediate views, we can flexibly place the views randomly in the grid by specifying their row and column positions. More than one view can be placed in a grid cell. Besides this, views can span multiple grid cells too.

NOTE

No need to specify `layout_height` and `layout_width` for the GridLayout child views as they default to `WRAP_CONTENT`.

Specifying Row and Column Position

The two attributes that are used to specify the row and column position of the grid cell for inserting views are `android:layout_row` and `android:layout_column`. Together, they specify the exact location of the grid cell for placing the view. For example, the following statements place the view at the first row and column position of the grid:

```
android:layout_row="0"
android:layout_column="0"
```

When either or both of the preceding attributes are not specified, GridLayout uses the next grid cell by default for placing the view.

Spanning Rows and Columns

Views can span rows or columns if desired. The attributes used for doing so are `android:layout_rowSpan` and `android:layout_columnSpan`. For example, the following statement spans the view to two rows:

```
android:layout_rowSpan="2"
```

Similarly, the following statement spans the view to three columns:

```
android:layout_columnSpan="3"
```

Inserting Spaces in the GridLayout

For inserting spaces, a spacing view called Space is used. That is, to insert spaces, the Space view is inserted as a child view. For example, the following statements insert a space at the second row in the GridLayout. The width and height of the blank space are 50dp and 10dp:

```
<Space
    android:layout_row="1"
    android:layout_column="0"
    android:layout_width="50dp"
    android:layout_height="10dp" />
```

Similarly, the following statements insert a space at the third row in the GridLayout that spans three columns:

```
<Space
android:layout_row="3"
```

```
android:layout_column="0"
android:layout_columnSpan="3"
android:layout_gravity="fill" />
```

Let's apply the knowledge gained so far in arranging controls in a GridLayout. The application has controls arranged in the same way as we saw in TableLayout (see Figure 3.14) but in GridLayout instead. So, let's create a new Android project called `GridLayoutLayoutApp`. Make its layout file, `activity_grid_layout_app.xml`, appear as shown in Listing 3.14.

LISTING 3.14 The Layout File `activity_grid_layout_app.xml` on Arranging Controls in a GridLayout Container

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:rowCount="7"
    android:columnCount="2" >
    <TextView
        android:layout_row="0"
        android:layout_column="0"
        android:text="New Product Form"
        android:typeface="serif"
        android:layout_columnSpan="2"
        android:layout_gravity="center_horizontal"
        android:textSize="20dip"  />
    <Space
        android:layout_row="1"
        android:layout_column="0"
        android:layout_width="50dp"
        android:layout_height="10dp" />
    <TextView
        android:layout_row="2"
        android:layout_column="0"
        android:text="Product Code:"  />
    <EditText
        android:id="@+id/prod_code"
        android:layout_width="100dip"  />
    <TextView
        android:text="Product Name:"   />
    <EditText
        android:layout_row="3"
        android:layout_column="1"
        android:id="@+id/prod_name"
        android:layout_width="200dip"   />
    <TextView
        android:layout_row="4"
        android:layout_column="0"
        android:text="Product Price:"   />
    <EditText
        android:layout_row="4"
        android:layout_column="1"
        android:id="@+id/prod_price"
        android:layout_width="100dip"  />
    <Space
        android:layout_row="5"
        android:layout_column="0"
        android:layout_width="50dp"
        android:layout_height="20dp" />
    <Button
        android:layout_row="6"
        android:layout_column="0"
        android:id="@+id/add_button"
        android:text="Add Product"   />
    <Button
        android:id="@+id/cancel_button"
        android:text="Cancel"   />
</GridLayout>
```

In the preceding code, the `GridLayout` is defined as consisting of seven rows and two columns. The orientation of `GridLayout` is set to horizontal; that is, controls are placed in rows. It means that while specifying the grid location of a view, if we don't specify the column number, the next available column is assigned to it. As said earlier, the `layout_width` and `layout_height` attributes are not specified for any of the views laid in GridLayout because the default value `wrap_content` is considered for them. Remember, the row and column numbers are zero-based. In Listing 3.14, the controls are positioned in the grid as follows:

- A `TextView` with the text `New Product Form` is set to appear at the first row and column position of the grid. The text appears in serif font and in `20dip` size. The text spans two columns and appears at the center of the row.
- A blank space is inserted at the second row and first column position. The width and height of the blank space are `50dp` and `10dp`, respectively.
- A `TextView` with the text `Product Code:` is set to appear at the third row and first column position of the grid.
- An `EditText` control with the ID `prod_code` of width `100dip` is set to appear at the third row and second column position of the grid, that is, to the right of the text `Product Code:`. The question is even though we didn't specify row and column position for the `EditText` control, how it will appear at the third row and second

column position? The answer is because the orientation of the `GridLayout` is horizontal, the current row (if it is not full) and the next column (if available) are considered the default location for the control to be inserted.

- A `TextView` with the text `Product Name:` is set to appear at the fourth row and first column position of the grid. Because both columns of the third row are full, the fourth row is considered the location for this view.
- An `EditText` control with the ID `prod_name` of width `200dip` is set to appear at the fourth row and second column of the grid, that is, to the right of the text `Product Name:.`
- A `TextView` with the text `Product Price`: is set to appear at the fifth row and first column of the grid.
- An `EditText` control with the ID `prod_price` of width `100dip` is set to appear at the fifth row and second column position of the grid, that is, to the right of the text `Product Price:.`
- A blank space is inserted at the sixth row and first column position. The width and height of the blank space are `50dp` and `20dp`, respectively.
- A `Button` control with the caption `"Add Product"` is set to appear at the seventh row and first column of the grid.
- A `Button` control with the caption `"Cancel"` is set to appear at the seventh row and second column of the grid.

There is no need to write any code in the Java activity file `GridLayoutAppActivity.java`. When the application is run, the controls are laid out in the grid pattern as shown in Figure 3.15.

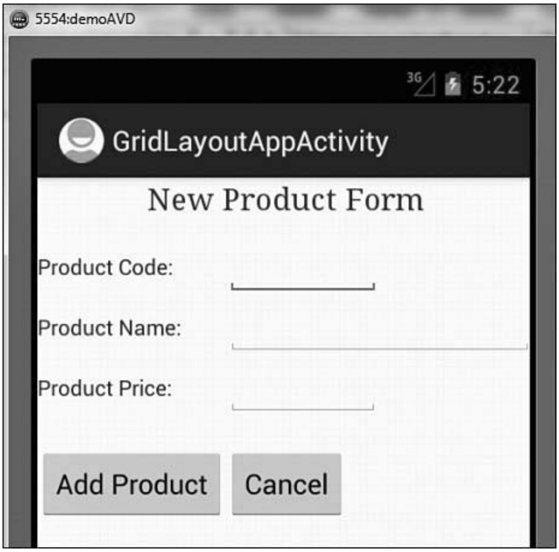


FIGURE 3.15 Controls organized in the GridLayout

Adapting to Screen Orientation

As with almost all smartphones, Android supports two screen orientations: `portrait` and `landscape`. When the screen orientation of an Android device is changed, the current activity being displayed is destroyed and re-created automatically to redraw its content in the new orientation. In other words, the `onCreate()` method of the activity is fired whenever there is a change in screen orientation.

`Portrait` mode is longer in height and smaller in width, whereas `landscape` mode is wider but smaller in height. Being wider, `landscape` mode has more empty space on the right side of the screen. At the same time, some of the controls don't appear because of the smaller height. Thus, controls need to be laid out differently in the two screen orientations because of the difference in the height and width of the two orientations.

There are two ways to handle changes in screen orientation:

- **Anchoring controls**—Set the controls to appear at the places relative to the four edges of the screen. When the screen orientation changes, the controls do not disappear but are rearranged relative to the four edges.
- **Defining layout for each mode**—A new layout file is defined for each of the two screen orientations. One has the controls arranged to suit the `Portrait` mode, and the other has the controls arranged to suit the `Landscape` mode.

Anchoring Controls

For anchoring controls relative to the four edges of the screen, we use a RelativeLayout container. Let's examine this method by creating an Android project called `ScreenOrientationApp`. To lay out the controls at locations relative to the four edges of the screen, write the code in the layout file `activity_screen_orientation_app.xml` as shown in Listing 3.15.

LISTING 3.15 The Layout file `activity_screen_orientation_app.xml` on Laying Out Controls Relative to the Four Edges of the Screen

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dip"
        android:layout_marginLeft="20dip" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="28dip"
        android:layout_toRightOf="@id/Apple"
        android:layout_marginLeft="15dip"
        android:layout_marginRight="10dip"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="200dip"
        android:layout_height="50dip"
        android:layout_marginTop="15dip"
        android:layout_below="@id/Apple"
        android:layout_alignParentLeft="true" />
    <Button
        android:id="@+id/Grapes"
        android:text="Grapes"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:minWidth="100dp"
        android:layout_alignParentRight="true"
        android:layout_below="@id/Banana" />
    <Button
        android:id="@+id/Kiwi"
        android:text="Kiwi"
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_below="@id/Banana"
        android:paddingTop="15dip"
        android:paddingLeft="25dip"
        android:paddingRight="25dip" />
</RelativeLayout>
```

Listing 3.15 shows five `Button` controls arranged in a RelativeLayout container. The controls are aligned relative to the edges of the container or in relation to each other. Let's keep the activity file `ScreenOrientationAppActivity.java` unchanged with the default code, as shown in Listing 3.16.

LISTING 3.16 Default Code in the Java Activity File `ScreenOrientationAppActivity.java`

```
package com.androidunleashed.screenorientationapp;

import android.app.Activity;
import android.os.Bundle;
```

```
public class ScreenOrientationAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_screen_orientation_app);
    }
}
```

When the application is run while in the default `portrait` mode, the controls appear as shown in Figure 3.16 (left). Because the five `Button` controls are placed in relation to the four edges of the container and in relation to each other, none of the `Button` controls disappear if the screen is rotated to `landscape` mode, as shown in Figure 3.16 (right). To switch between `portrait` mode and `landscape` mode on the device emulator, press the `Ctrl+F11` keys.

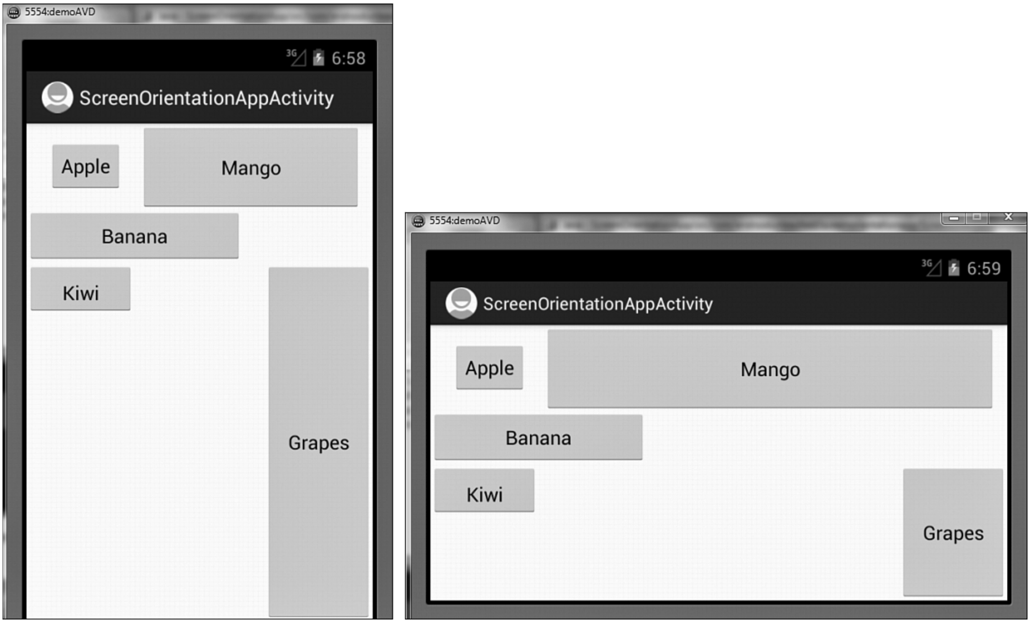


FIGURE 3.16 (left) Controls in `portrait` mode, and (right) the controls in `landscape` mode

Now that we understand the concept of adapting to screen orientation through anchoring controls, let's have a look at another approach.

Defining Layout for Each Mode

In this method, we define two layouts. One arranges the controls in the default `portrait` mode, and the other arranges the controls in `landscape` mode. To understand this, let's write code as shown in Listing 3.17 for laying out the controls for `portrait` mode in the default layout file `activity_screen_orientation_app.xml` (found in the `res/layout` folder).

LISTING 3.17 The Layout File `activity_screen_orientation_app.xml` on Laying Out Controls in `portrait` Mode

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="300dip"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"  />
    <Button
        android:id="@+id/Grapes"
        android:text="Grapes"
        android:layout_width="300dip"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"   />
    <Button
        android:id="@+id/Kiwi"
        android:text="Kiwi"
        android:layout_width="300dip"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"  />
</LinearLayout>
```

In Listing 3.17, we can see that five `Button` controls are vertically arranged in a LinearLayout container, one below the other. This vertical arrangement makes a few of the `Button` controls disappear when the screen is in `landscape` mode.

If we run the application without defining the layout for the `landscape` mode, we find the controls arranged in `portrait` mode, as shown in Figure 3.17 (left). But when we switch the screen orientation to `landscape`, we find the last two `Button` controls have disappeared, as shown in Figure 3.17 (right). This is because in `landscape` mode, the screen becomes wider but shorter in height.

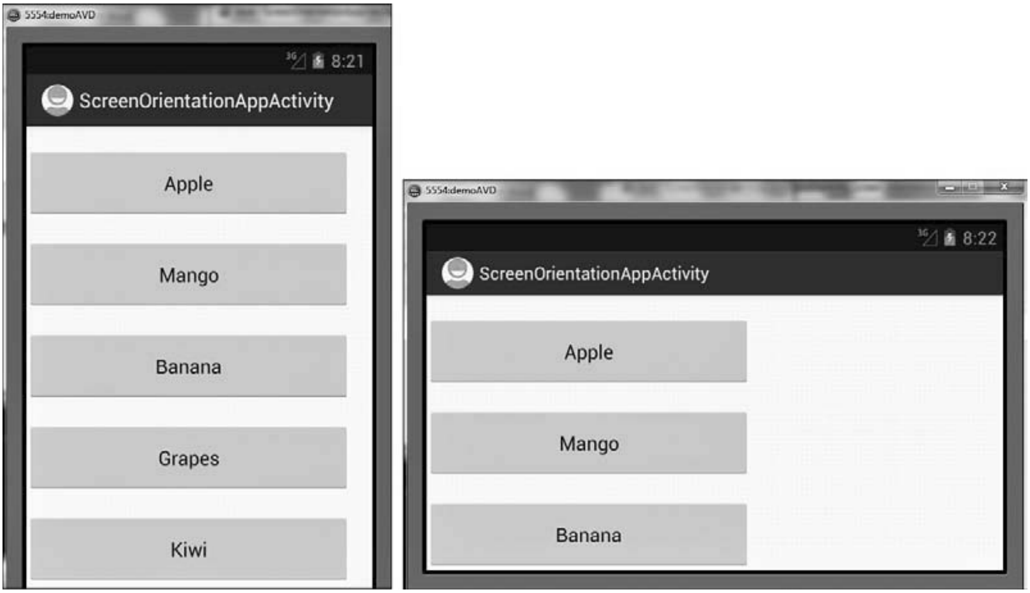


FIGURE 3.17 (left) Controls in `portrait` mode, and (right) some controls disappear in `landscape` mode.

To use the blank space on the right side of the screen in `landscape` mode, we need to define another layout file, `activity_screen_orientation_app.xml`, created in the `res/layout-land` folder. The `layout-land` folder has to be created manually inside the `res` folder. Right-click on the `res` folder in the `Package Explorer` window and select the `New`, `Folder` option. A dialog box opens, asking for the name for the new folder. Assign the name `layout-land` to the new folder, and click the `Finish` button. Copy the `activity_screen_orientation_app.xml` file from the `res/layout` folder and paste it into `res/layout-land` folder. Modify the `activity_screen_orientation_app.xml` file in the `res/layout-land` folder so as to arrange the controls in `landscape` mode. The code in the newly created `activity_screen_orientation_app.xml` is modified as shown in Listing 3.18.

LISTING 3.18 The Layout File `activity_screen_orientation_app.xml` in the `res/layout-land` Folder

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"
        android:layout_toRightOf="@id/Apple" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="250dip"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"
        android:layout_below="@id/Apple" />
    <Button
        android:id="@+id/Grapes"
        android:text="Grapes"
        android:layout_width="250dip"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"
        android:layout_below="@id/Apple"
        android:layout_toRightOf="@id/Banana"  />
    <Button
        android:id="@+id/Kiwi"
        android:text="Kiwi"
        android:layout_width="250dip"
        android:layout_height="wrap_content"
        android:padding="20dip"
        android:layout_marginTop="20dip"
        android:layout_below="@id/Banana" />
</RelativeLayout>
```

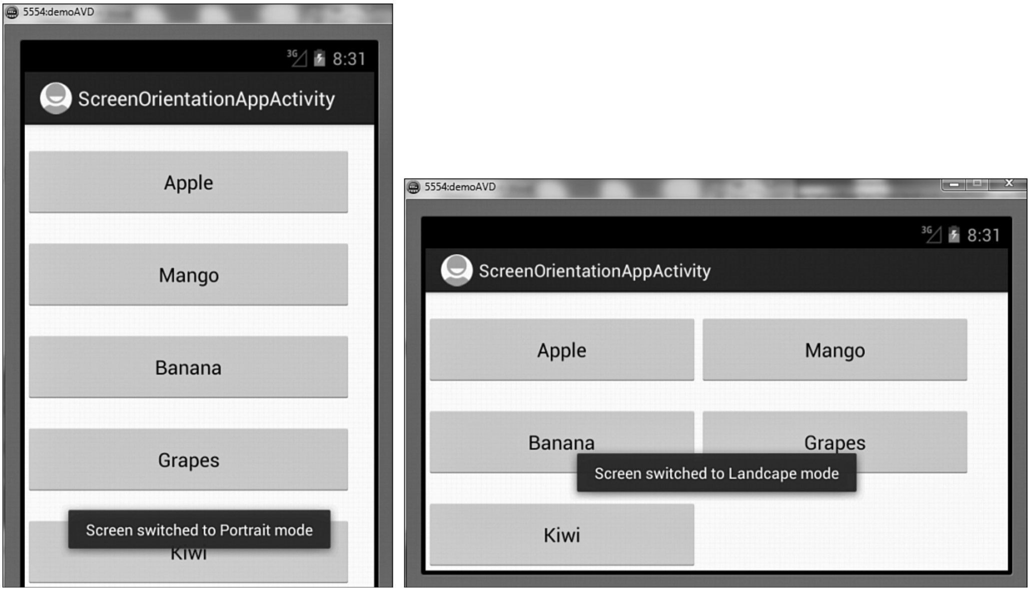
In this code block, we can see that, to fill up the blank space on the right side of the screen, the `Mango` and `Grapes` button controls are set to appear to the right of the `Apple` and `Banana` button controls.

We can also detect the screen orientation via Java code. Let's modify the activity file `ScreenOrientationAppActivity.java` to display a toast message when the screen switches between `landscape` mode and `portrait` mode. The code written in the Java activity file `ScreenOrientationappActivity.java` is shown in Listing 3.19.

LISTING 3.19 Code Written in the Java Activity File `ScreenOrientationappActivity.java`

```
package com.androidunleashed.screenorientationapp;

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

public class ScreenOrientationAppActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_screen_orientation_app);
   if(getResources().getDisplayMetrics().widthPixels>getResources().getDisplayMetrics().
            heightPixels)
        {
           Toast.makeText(this,"Screen switched to Landscape mode",Toast.LENGTH_SHORT).
           show();
        }
        else
        {
           Toast.makeText(this,"Screen switched to Portrait mode",Toast.LENGTH_SHORT).
show();
        }
    }
}
```

Now, when we run the application, the controls appear in `portrait` mode as shown in Figure 3.18 (left) and in `landscape` mode as shown in Figure 3.18 (right). We can see that none of the `Button` controls are now hidden in `landscape` mode.



Screen switched to Portrait mode

Screen switched to Landcape mode

FIGURE 3.18 (left) Controls in `portrait` mode, and (right) all controls are visible in `landscape` mode.

Summary

In this chapter, you learned how to lay out controls for different orientations. You also learned to apply attributes such as `Orientation`, `Height`, `Width`, `Padding`, `Weight`, and `Gravity` to arrange the controls and their content. You saw how to create individual Android applications dedicated to each layout, `LinearLayout`, `RelativeLayout`, `AbsoluteLayout`, `FrameLayout`, and `TableLayout`.

In the next chapter, you learn about different types of resources and the procedures to apply them in Android applications. You learn to apply `Dimension` resources, `Color` resources, `Styles`, and `Themes` and also learn to use `String` and `Integer` arrays. To display images in the Android application, you learn to use `Drawable` resources and create an Image Switcher application using the `ToggleButton` control.

This page intentionally left blank

Index

| (pipe character) operator, 43, 108

A

above attribute, 113

AbsoluteLayout, 30

- arrangements, 123
- controls, 121-123
- description, 66, 102

AccelerateDecelerateInterpolator, 471

AccelerateInterpolator, 471

ACCELEROMETER sensor type, 626, 628-630

AccessSync class, 601

<action> tags, 63

Action Views, 359, 362, 369

ActionBar. *See also* drop-down list ActionBar; tabbed ActionBar

- action items, 359
  - displaying, 362-369
  - icons, 372
- Action Views, 359, 362, 369
- attributes
  - alphabeticShortcut, 375
  - icon, 361
  - logo, 361
  - minSdkVersion, 377
  - numericShortcut, 375
  - showAsAction, 362, 371, 375
- components, 359
- enabling, 360-361
- features, 359
- Menu Items, 358-359

methods
getActionBar(), 360, 366
onCreateOptionsMenu(), 372
onOptionsItemSelected(), 361-362, 366
setDisplayShowHomeEnabled(), 361
setDisplayShowTitleEnabled(), 361
setHomeButtonEnabled(), 361, 366
show() and hide(), 360
navigating with application icons, 361-362
Options Menus, 359
Overflow Menu, 359, 373
replacing menus, 370-377

Activity class/activities
basics, 59
creating, 29, 79-80
definition, 30
event handling, 68, 71-72
initializing, 34
Java code, 34-35
life cycles, 60
main application file, 58-59
methods
dismissDialog(), 260
onClick(), 68-69, 72
onCreate(), 34, 60, 69
onCreateDialog(), 260-261
onCreateOptionsMenu(), 35
onDestroy(), 60
onPause(), 60
onPrepareDialog(), 260
onResume(), 60
onStart(), 60
onStop(), 60
removeDialog(), 260
setContent(), 34
showDialog(), 260-261
startActivity(), 77-78
registering, 80-81
starting, 77-78, 81-82

<activity> tags, 63

Adapters
ArrayAdapter
AutoCompleteText control, 226-227
ListView control, 214-217
setListAdapter() method, 304
Spinner control, 223-225
CursorAdapter, ListView control, 214
ImageAdapter (custom), GridView control, 234-235
ListAdapter, ListView control, 211, 214
PageAdapter, 236

ADB (Android Debug Bridge)
accessing, 394
commands, 48-49
application package's databases directory, 395-396
lists of devices/emulators, 394
lists of directories and files in data directory, 395
lists of directories in emulators/devices, 395
components, 48
restarting, 18
SQLite commands
.exit, 397
.schema, 396
SQL DELETE, 397
SQL SELECT, 396-397
SQL UPDATE, 397
.tables, 396
uses, 48, 394

add() method, parameters, 345

addPreferencesFromResource() method, 315

addRow() method, 390, 393, 405, 408

addSubMenu() method, 346

addTab() method, 377

addToBackStack() method, 296

adjustViewBounds attribute, 125

ADT (Android Development Tools) plug-in

- ADT Wizard, /res folder, 147
- Android DDMS (Dalvik Debug Monitor Server), 22
- Android Development Tools, 22
- Android Hierarchy Viewer, 22
- Android Traceview, 22
- application files/directories/subdirectories, 56
- attaching ADT to Eclipse IDE, 24
- downloading, 12
- installing, 22
- license agreement, 23
- software updates, 24
- uses, 19-20

AlarmManager

- alarms
  - ELAPSED_REALTIME, 619
  - ELAPSED_REALTIME_WAKEUP, 620
  - repeating, 620, 625
  - RTC, 619
  - RTC_WAKEUP, 619
  - for specific date and time, 620-625
- definition, 619
- methods
  - cancel(), 620
  - getSystemService(), 619
  - set(), 619, 625
  - setInexactRepeating(), 620
  - setRepeating(), 620, 625
  - setTimeInMillis, 621

AlertDialog

- alertDialog object, 261, 263
- AlertDialog.Builder subclass, 261
  - methods, 261-262
- Builder object, 262-263
- user input, 263-267

AlertDialog.Builder subclass, 261

- methods
  - setIcon, 261
  - setMessage, 261
  - setNegativeButton, 261
  - setNeutralButton, 261
  - setPositiveButton, 261
  - setTitle, 261

alignBaseline attribute, 114

alignBottom attribute, 114

alignLeft attribute, 114

alignParentBottom attribute, 113

alignParentLeft attribute, 113

alignParentRight attribute, 113

alignParentTop attribute, 113

alignRight attribute, 114

alignTop attribute, 113

Allocation Tracker tab, DDMS, 242

alpha animations, 453, 455-456

AlphaAnimation class, 464-465

alphabeticShortcut attribute, 331, 375

always value, 362

alwaysScroll value, 209

AMBIENT_TEMPERATURE sensor type, 626

Android 1.0 (API 1) through 4.1 (API 16) packages, 16, 55

android attribute, 62

Android Debug Bridge. *See* ADB

Android Developer Console, 639-640

How can we make this index more useful? Email us at indexes@samspublishing.com

- Android emulator
  - commands, 47
  - DDMS (Dalvik Debug Monitor Service) perspective, 48
  - limitations, 47, 628
  - uses, 47
- Android Hierarchy Viewer, 22
- Android Manifest Editor, 64
- Android Manifest file, 58
  - <action> tag, 63
  - activities
    - <activity> tag, 63
    - defining, 60-61
    - registering, 80-81
    - starting, 81
  - applications
    - <application>, attributes, 62
    - registering content providers, 587-588
    - running, 39
  - <category>, 63
  - default code, 61-64, 163
  - intents
- defining, 61
- <intent-filter> tag, 63
  - <manifest>, attributes, 62
  - notifications, 530
  - Overview screen, 64
  - permissions
    - Contacts Provider, 565
    - defining, 60
    - Internet, 601
    - SMS messages, receiving, 543-544
    - SMS messages, sending, 534
    - telephony services, 554, 556-557
  - <provider>, 63
  - receivers
    - <receiver> tag, 63
    - registering broadcast receivers, 519, 521-522
  - services
    - defining, 60
    - <service> tag, 63
  - theme attribute, 163-164
  - <uses-sdk>, attributes, 62
  - versioning tags, 633-634
    - <uses-configuration>, 636
    - <uses-feature>, 636
    - <uses-permissions>, 63-64, 554, 636
- Android Platform SDK Starter Package, 12
- Android projects
  - creating, 27
    - Android Project Creator, 27, 53
    - Android Project Wizard, 28, 53
  - launching, 35
  - naming, 27
  - package names, 27
  - target platforms, 27, 53
  - workspace location, 28, 54
- Android Runtime
  - core Android libraries, 12
  - core Java libraries, 12
  - Dalvik Virtual Machine, 12
- Android SDK. *See* SDK (Software Development Kit)
- Android software stack, 11
  - Android Runtime
    - core Android libraries, 12
    - core Java libraries, 12
    - Dalvik Virtual Machine, 12
  - application framework, classes, 12
  - application layer, built-in and developed, 12
  - libraries, 11
    - FreeType, 11

media, 11
SQLite, 11
Surface Manager, 11
Linux kernel, 11
Android Traceview, 22
anim subdirectory, 149
animateTo(), 500
AnimationDrawable class, 449-453
animations. *See also* frame-by-frame animations; tweening animations
collecting and sequencing, 466-470
loading, 460-462
Property animations, 445
repeating, 463-464
View animations
frame-by-frame animations, 446-453
tweening animations, with XML, 454-455
anonymous class, 68-71
antialiasing, 425
AnticipateInterpolator, 471
AnticipateOvershootInterpolator, 471
antipiracy support, 11
APIs (Application Programming Interfaces)
code names, 55
platforms and API levels, 55-56
<uses-sdk> tags, attributes, 62
values-v11 and v12 folders, 149
APK files
digital certificates, 638
uploading to Google Play Store, 640
applications
Activity class/activities
basics, 59
creating, 29, 54
definition, 30
life cycles, 60
main application file, 58-59
<application> tags, attributes, 62
backward compatibility, 29, 56
digital certificates, Export Android Application Wizard, 637-638
files, 28
created by ADT plug-in, 29, 31
Java, 30
XML, 30
folders/files/directories/subdirectories, 56-58
Google Play Store
Android Developer Console, 639-640
application distribution, for free, 640, 642
application distribution, for price, 640, 642
application distribution, with embedded advertisements, 642
developer accounts, 639-641
launching, 35
debug configurations, 35, 50
Eclipse launch configuration, 69
on handsets, 50
run configuration, 35, 50
naming/renaming, 27, 35, 54
screens
boot screens, 36
Home screen, unlocking, 36-37
sizing, 38
target platforms, 54-56
threads. *See* background threads
title name, 29
versioning information, 633-635
Arabic language support, 10
arcs, 434-435
array resources, 165. *See also* string resources
integer arrays, 149-170
string arrays, 165-168
populating Spinner control, 221-222

ArrayAdapter
  AutoCompleteText control, 226-227
  ListView control, 214-217
  setListAdapter() method, 304
  Spinner control, 223-225
AssetManager, 204-206
assets folder/directory, 57
  AssetManager, 204-206
  info.txt in TextView control, 205-206
  InputStream class, available() method, 206
  methods
    getAssets(), 206
    open(), 206
versus res folder/resources, 204
AsyncTask class, 591, 594-597
  asynchronous processing, 591
  methods
    doInBackground(), 594-597, 600
    onCancelled(), 595
    onPostExecute(), 595, 601
    onPreExecute(), 595, 597
    onProgressUpdate(), 594-597
    publishProgress(), 594, 597
Audio device driver, 11
audio files
  MediaPlayer methods
    create(), 189, 191
    pause(), 191
    start(), 189, 191
  ProgressBar control, 201
  raw subdirectory, 187
AutoCompleteTextView controls
  ArrayAdapter, 226-227
  definition, 225
  methods
    setAdapter(), 227
    setThreshold(), 227
autoSize attribute, 83
available() method, 206
AVD (Android Virtual Device), 24-25
  AVD Manager dialog, 25-26
  DDMS, 240, 245
  devices
    creating, 25-26
    specifications, 25-26
  map location applications, 485-486
  scaling to real device size, 38
  SMS messaging
    receiving, 545
    sending, 539-540
  targets, selecting, 36-35

B

background attribute, 45
  ToggleButton control, 176-179
background threads
  AsyncTask class, 591, 594-597
    asynchronous processing, 591
    methods, 594-595, 597
  Handler class, 591-594
    methods, 592, 594
below attribute, 113
/bin folder, 57-58
bindService() method, 608, 614, 616
BitmapFactory class, 601
bitmaps, 441-443
Bluetooth with Simple Secure Pairing, 10
Boolean values, 274
Bornstein, Dan, 38
bottom value, 43, 103
BounceInterpolator, 471

bound services
- definition, 607
- IncomingHandler object, 616
- methods
  - bindService(), 614, 616
  - onBind(), 614
  - onServiceConnected(), 616, 618
  - onServiceDisconnected(), 616, 618
- ServiceConnection interface, 616, 618

breakpoints, 245-246

Breakpoints pane, Debug perspective, 247, 250
- buttons
  - Collapse All, 252
  - Expand All, 252
  - Go to File for Breakpoint, 252
  - Link with Debug View, 252
  - Remove All Breakpoints, 252
  - Remove Selected Breakpoints, 252
  - Show Breakpoints Supported by Selected Target, 252
  - Skip All Breakpoints, 252
- properties, 250-251

BroadcastReceiver class, 519, 521-522
- methods
  - getAction(), 519
  - getResultCode(), 538
  - getStringExtra(), 519
  - onReceive(), 519, 521
- registering, 538, 543

Browser content provider, 559

Builder object, 262-263

Bundle object, 614, 618
- getExtras() method, 613

Button controls, 30, 66
- animations
  - loading, 460-462
  - tweening, 454-455
- arrangements
  - AbsoluteLayouts, 123
  - GridLayouts, 135-137
  - LinearLayouts, 104-111
  - RelativeLayouts, 115-117, 119-120
  - TableLayouts, 131-133
- attributes
  - drawableBottom, 189
  - drawableLeft, 189
  - drawableRight, 189
  - drawableTop, 189
- Click Me, 70
- definition, 67
- EditText control, 67, 70
- methods
  - dispMessage(), 72-74
  - onClick(), 68-69, 72
- screen orientation, 138-144

C

Calendar instance
- alarms, 621
- DatePickerDialog, 267, 270
- TimePickerDialog, 274

callExcep() method, 253

CallLog content provider, 559

Camera app, 10

Camera device driver, 11

cancel() method, 620

canGoBack() method, 478

canGoBackOrForward() method, 477-478

canGoForward() method, 477-478

Canvas and Paint objects, 421-423
- colors, 423-424
- displaying text, 443-444

drawing
arcs, 434-435
bitmaps, 441-443
canvas height and width, 424-425
circles, 424
defining drawing paths, 429-430
lines, 427
ovals, 433-434
points, 425
rectangles, 430-431
rectangles, rounded, 432-433
gradients, 436
LinearGradient, 436-438
RadialGradient, 439
SweepGradient, 440
paint antialiasing, 425
paint styles, 425
strokes
cap, 426
width, 426
canvas height and width, 424-425
capitalize attribute, 83
<category> tags, 63
Cause GC (Garbage Collection), Devices tab, DDMS, 241
C/C++ libraries, Android software stack, 11
cells, TableRow object, 129, 132
center value, 43, 125, 190
centerHorizontal attribute, 33, 43, 113
centerInParent attribute, 113
centerInside value, 125
centerVertical attribute, 33, 40, 113
character value, 83
check boxes
Context Menus, 337
Options Menus, 331
Submenus, 347-348, 375
checkableBehavior attribute, 334, 337
CheckBox controls
checked/unchecked states, 87-88, 90-91
definition, 67
methods
onCheckedChanged(), 87
onClick(), 87, 90
setChecked(), 87
toggle(), 87-91
CheckBoxPreference View, 312-313, 316-318
getBoolean() method, 317
choiceMode attribute, 209
circles, 424
clearCache() method, 478
clearHistory() method, 478
clickable attribute, 488
client-server program. *See* ADB (Android Debug Bridge)
close() method, 385
collapseColumns attribute, 130
color resources, 149, 156-158
Alpha channel, 156-157
hexadecimal RGB values, 156
color subdirectory, 149
colors, 423-424
column attribute, 134
columns, 134-135
TableLayout, 129-131
columnSpan attribute, 134
columnWidth attribute, 229-231
commit() method, 295-296
communication improvements
Bluetooth with Simple Secure Pairing, 10
Near Field Communication, 10
Console, DDMS, 245, 248
Contacts content provider, 559
data
accessing and using, 562-565
entering, 561

containers/layouts
attributes
above, 113
alignBaseline, 114
alignBottom, 114
alignLeft, 114
alignParentBottom, 113
alignParentLeft, 113
alignParentRight, 113
alignParentTop, 113
alignRight, 114
alignTop, 113
below, 113
centerHorizontal, 33, 113
centerInParent, 113
centerVertical, 33, 40, 113
column, 134
columnSpan, 134
gravity, 101, 109-110, 176-177
height, 33, 83, 101
margin, 114
marginBottom, 114
marginLeft, 114
marginRight, 114
marginTop, 114
row, 134
rowSpan, 134
span, 131
toLeftOf, 113
toRightOf, 113
weight, 101, 107, 111
width, 33, 82, 101
content alignment, 43
creating, 78-79
dimensions, units of measurement, 42-43, 83, 123
Views and ViewGroups, 30, 33, 101

content providers
CallLog, 559
characteristics of, 559
Contacts, 559
data, accessing and using, 562-565
data, entering, 561
custom providers
defining, 566-568
defining databases, 568-569
defining MIME types, 570-571
loaders, 581-583
methods, 571-581
steps, 566
URIs (Universal Resource Identifiers), 569-570
Media Store, 559
Settings, 559
STD (subscriber trunk dialing)
data display, 584-585
data entry, 566-568, 580, 588-589
data retrieval, 569-570, 587
data storage, 568
URIs (Universal Resource Identifiers), 560-561
authority, 560
date path, 560
id, 560
standard prefixes, 560
ContentResolver, 570, 587
methods
delete(), 587
notifyChange(), 587
Context Menus
applying to ListView control, 354-357
creating by coding, 349-353
onCreateContextMenu() method, 353

creating in XML, 336-344

checkableBehavior attribute, <group> node, 337

definition, 324

isChecked() method, 340

onContextItemSelected() method, 340

onCreateContextMenu() method, 338-340

registerForContextMenu() method, 338-339

setHeaderIcon() method, 340

setHeaderTitle() method, 340

CountriesProvider class, 568

create() method, 189, 191, 203, 445

createChooser() method, 546

createFromPdu() method, 542

createItem() method, 514

Cupcake code name, platform and API levels, 55

Cursor class, 392-394

methods

delete(), 587

getColumnIndexOrThrow(), 392

getColumnName(), 392

getColumnNames(), 392

getCount(), 392

getPosition(), 392

moveToFirst(), 392

moveToNext(), 392

moveToPosition(), 392

moveToPrevious(), 392

notifyChange(), 587

CursorAdapter, 214

CursorLoaders class, 581-583

custom content providers

defining, 566-568

databases, 568-569

MIME types, 570-571

loaders

CursorLoaders class, 581-583

getLoaderManager() method, 583

initLoader() method, 583

LoaderManager class, 581-583

onCreateLoader() method, 583

onLoaderReset() method, 583

onLoadFinished() method, 583

methods, 575-578

delete(), 574-575

getContentResolver(), 570

getType(), 571

insert(), 573, 578-580

query(), 572, 587

update(), 573-574

withAppendedId(), 573

STD (subscriber trunk dialing)

data display, 584-585

data entry, 566-568, 580, 588-589

data retrieval, 569-570, 587

data storage, 568

steps, 566

URIs (Universal Resource Identifiers), 569-570

CycleInterpolator, 471

D

Dalvik Debug Monitor Server. *See* DDMS

Dalvik Virtual Machine, 12, 38

data entry forms

creating, 401

rows, displaying, 410

DatabaseManager class, 386-390, 392

methods

addRow(), 393

retrieveRows(), 390, 393

SQLHelper class, 390

SQLiteOpenHelper class, 390

databases. *See also* SQLite relational databases

accessing

with ADB, 394-397

through menus, 398-401

date value, 84

DatePickerDialog, 259, 268-270

Calendar instance, 267, 270

onDateSet() method, 270, 278

datetime value, 84

DDMS (Dalvik Debug Monitor Server), 22

Allocation Tracker tab, 242

Android emulator, 48

Console tab, 245

devices or AVD, 240, 245

Devices tab, 240

Cause GC (Garbage Collection), 241

Debug, 240

Dump HPROF file, 241

Screen Capture/options, 241-242

Start Method Profiling, 241

Stop Process, 241

Update Heap, 240

Update Threads, 241

Emulator Control tab, 243-244

latitude and longitude values, 494

File Explorer tab, 243-242

Heap tab, 242

LogCat tab, 245

Network Statistics tab, 243

Threads tab, 242

Name, 242

Status, 242

stime, 242

Thread ID, 242

utime, 242

uses, 239-240

video files, loading onto SD cards, 195

Debug pane, Debug perspective, 247-248

buttons

Disconnect, 248

Drop To Frame, 248

Remove All Terminated Launches, 248

Resume, 248

Step Into, 248

Step Over, 248

Step Return, 248

Suspend, 248

Terminate, 248

Use Step Filters, 248-249

Debug perspective, DDMS, 48

panes

Breakpoints, 247, 250-254

Console, 248

Debug, 247-249

Editor, 247

Expressions, 249

LogCat, 248, 255-256

Outline, 248

Variables, 247, 254

switching from Java, 246

DecelerateInterpolator, 471

decodeStream() method, 601

delete() method, 574-575

Deployment Target Selection Mode options, 35

destroyItem() method, 239

Devices tab, DDMS, 240

Cause GC (Garbage Collection), 241

Debug, 240

Dump HPROF file, 241

Screen Capture/options, 241-242

- Start Method Profiling, 241
- Stop Process, 241
- Update Heap, 240
- Update Threads, 241

dex format, 38

dialog windows
- Activity class methods
  - dismissDialog(), 260
  - onCreateDialog(), 260-261
  - onPrepareDialog(), 260
  - removeDialog(), 260
  - showDialog(), 260-261
- AlertDialog, 259
  - alertDialog object, 261, 263
  - AlertDialog.Builder subclass, 261-262
  - Builder object, 262-263
  - user input, 263-267
- CharacterPickerDialog, 259
- DatePickerDialog, 259, 268-270
  - Calendar instance, 267, 270
  - onDateSet() method, 270, 278
  - with TimePickerDialog, 275-281
- Dialog, 259
- DialogFragment with FragmentManager, 260
- modal dialogs, 260
- ProgressDialog, 260
- TimePickerDialog, 260, 271-272
  - Boolean values, 274
  - Calendar instance, 274
  - with DatePickerDialog, 275-281
  - onTimeSet() method, 274, 278
  - tasks performed, 272-273

DialogFragment, 305-311. *See also* FragmentManager; fragments; ListFragment; PreferenceFragment
- asynchronous, 305
- methods
  - NegativeButton(), 309-310
  - newInstance(), 308
  - onCreateDialog(), 309
  - onCreateView() and LayoutInflator object, 307
  - PositiveButton(), 309-310
  - show(), 310

digital certificates, 637-638

dimensions for controls/layouts
- getDimension() method, 155
- units of measurement, 42-43, 83, 123

dimens.xml file, 154

dip or dp (device-independent pixels) unit of measurement, 42, 43, 83

disabled value, 209

dismissDialog() method, 260

Display device driver, 11

dispMessage() method, 72-74

doInBackground() method, 594-595, 597, 601

Donut code name, platform and API levels, 55

downloadImage() method, 601

dp or dip (device-independent pixels) unit of measurement, 42-43, 83, 154

drawable attribute, 448

drawable resources, 58, 148
- formats supported, 170
- -hdpi subdirectory, 58, 147-149
- images, adding, 177-178, 236, 370
- -ldpi subdirectory, 58, 147-149
- -mdpi subdirectory, 58, 147-149
- referencing files, 171
- screen resolutions, 170, 183
- -xhdpi subdirectory, 58, 148-149

drawableBottom attribute, 189

drawableLeft attribute, 189

drawableRight attribute, 189

drawableTop attribute, 189

drawArc() method, 434-435
drawBitmap() method, 441-443
drawCircle() method, 423-424
drawing
 arcs, 434-435
 bitmaps, 441-443
 canvas height and width, 424-425
 circles, 424
 defining drawing paths, 429-430
 lines, 427
 ovals, 433-434
 points, 425
 rectangles, 430-431
  rounded, 432-433
drawLine() method, 427
drawOval() method, 433-434
drawPath() method, 429-430
drawPoint() method, 425
drawRect() method, 430-431
drawRoundRect() method, 432-433
drawSelectorOnTop attribute, 209, 212
drawText() method, 443-444
drop-down list ActionBar, 380-383
 ArrayAdapter, 380-381
 attributes
  minSdkVersion, 382
  theme, 382
 methods
  setListNavigationCallbacks(), 380-381
  setNavigationMode(), 380
 onNavigationItemSelected, 381-382
 onNavigationListener, 380
 SpinnerAdapter interface, 380-382
Dump HPROF file, Devices tab, DDMS, 241
duration attribute, 448

E

Eclair code name, platform and API levels, 55
Eclipse IDE (integrated development environment)
 downloading, 12
 installing, 20
 launch configuration, 35
  debug configuration, 35
  run configuration, 35
 uses, 19
 Welcome screen, 20
 Workbench, 21
 workspace locaton, 20
Editor pane, Debug perspective, 247
EditText controls, 30, 66
 arrangements
  AbsoluteLayout, 123
  GridLayout, 135-137
  RelativeLayout, 117-121
  TableLayout, 131-133
 attributes, 82-84
 Button control, 70, 84
 definition, 67
 event listeners, 84-86
 onKey() method, 476
EditTextPreference View, 313, 316-318
ELAPSED_REALTIME alarm, 619
ELAPSED_REALTIME_WAKEUP alarm, 620
ellipsize attribute, 45
email, 546-552
Emulator Control tab, DDMS, 243-244
enabled attribute, 488
entries attribute, 209, 212, 222-223
events/event handling
 anonymous class, 68-71

interfaces
OnCheckedChangeListener, 87
OnClickListener, 68, 71-72, 90
radioListener1 and radioListener2, 97
listeners, 67-69, 84-86
XML files, 72-75
execSQL() method, 390
execute() method, 601
.exit ADB command, 397
Export Android Application Wizard, 637-638
Expressions pane, Debug perspective, 249

F

Face Unlock, 10
false value, 209
File Explorer, DDMS, 195-197, 243-242
fillAfter attribute, 458
fill_horizontal value, 108
fill_parent constant, 33
findFragmentById() method, 295-296
findFragmentByTag() method, 295
findViewById() method, 42
fitCenter value, 125
fitXY value, 125
forward geocoding, 502-506
FragmentManager. *See also* DialogFragment; fragments; ListFragment; PreferenceFragment
communicating with Fragment1Activity, 296
Fragment class, 296
FragmentTransaction, 294
methods
addToBackStack(), 296
beginTransaction(), 294
commit(), 295-296
findFragmentById(), 295-296
findFragmentByTag(), 295
getArguments(), 296-297
getFragmentManager(), 294
remove(), 295
replace(), 295
setArguments(), 296-297
fragments. *See also* DialogFragment; FragmentManager; ListFragment; PreferenceFragment
communicating between, 296
life cycle, 282-283
methods
onActivityCreated(), 283
onAttach, 283
onCreate(), 283
onCreateView(), 283, 285-286
onDestroy(), 283
onDestroyView(), 283
onDetach(), 283
onPause(), 283
onResume(), 283
onStart(), 283
onStop(), 283
navigating to previous, 296-297
retrieving content, 297
states
onRestoreInstanceState, 297-298
onSaveInstanceState, 297-298
saving/restoring, 297-301
structure, 282
frame-by-frame animations, 446
attributes, 448
defining with Java code, 451-453
defining with XML, 446-451
View animations, 446-453
FrameLayout, 30
description, 67, 102, 125
ImageView controls, 83, 126-128

FreeType library, 11

fromAlpha attribute, 455

fromDegrees attribute, 456

fromPixel() method, 511

fromXDelta attribute, 458

fromXScale attribute, 457

fromYDelta attribute, 458

fromYScale attribute, 457

Froya code name, platform and API levels, 55

full_horizontal value, gravity attribute, 43

full_vertical value, gravity attribute, 43, 108

G

/gen folder, 57

geocoding

- forward geocoding, 502-506
- reverse geocoding, 502-503

Gesture Mode combined with voice, 10

getAction() method, 519

getActionBar() method, 360, 366, 379

getActivity() method, 525

getAddress() method, 506

getArguments() method, 296-297

getAssets() method, 206

getBoolean() method, 317

getCenter() method, 515

getColor() method, 158

getColumnIndexOrThrow() method, 392

getColumnName() method, 392

getColumnNames() method, 392

getContentResolver() method, 570

getCount() method, 235, 239, 392

getDefault() method, 537

getDefaultSensor() method, 627

getDimension() method, 155

getDrawable()method, 178, 194

getExtras() method, 292, 613

getFromLocation() method, 503-506

getFromLocationName() method, 503

getHeight() method, 424-425

getIntrinsicWidth() method, 515

getItem() method, 235

getItemId() method, 235

getLoaderManager() method, 583

getMessageBody() method, 543

getOriginatingAddress() method, 543

getOverlays() method, 510

getPosition() method, 392

getReadable() method, 390

getReadableDatabase() method, 385, 390

getResources() method, 155, 158, 178

getResultCode() method, 538

getSensorList() method, 627

getString() method, 317

getStringExtra() method, 519

getSystemService() method, 493, 527, 619, 626

getTimestampMillis() method, 543

getType() method, 571

getWidth() method, 424-425

getWritableDatabase() method, 385, 390

GIF files, 125, 170

Gingerbread code name, platform and API levels, 55

goBack() method, 477-478

goBackOrForward() method, 478

goForward() method, 477-478

Google Maps API

- AVD-based applications, 485-486
- displaying
  - map markers, 507-515
  - satellite view, 499

streets and places, 497
traffic view, 499, 501
Google Maps-based applications, 486-489
installing, 484-485
key, 483
applying for, 483
signing up for, 483-484
Google Now, 10
Google Play Store
application distribution
with embedded advertisements, 642
for free, 640, 642
for price, 640, 642
developer accounts, 639-641
Android Developer Console, 639-640
filters, versioning information, 635
Google Checkout Merchant Account, 640
widgets, 11
Google Search, 11
Google USB Driver package, 16
Google Voice search, 10
GPS Exchange Format (GPX), 495-496
GPX (GPS Exchange Format), 495-496
gradients, 436
LinearGradient, 436-438
RadialGradient, 439
SweepGradient, 440
graphics libraries, 11
gravity attribute, 43, 101, 109-110, 176-177
GRAVITY sensor type, 626
GridLayout
arrangements, 135-137
description, 33-67, 102, 133-134
rows and columns, operations on, 134-135
versus TableLayout, 133
GridView controls, 228
attributes
columnWidth, 229-231
horizontalSpacing, 229
numColumns, 228-229
stretchMode, values, 229
verticalSpacing, 229
definition, 227-228
image display, 231-235
ImageAdapter (custom), 234-235
GYROSCOPE sensor type, 626

H

handleMessage() method, 592, 594
Handler class, methods
handleMessage(), 592, 594
post(), 592
run(), 594
sendEmptyMessage(), 592
sendMessage(), 592
Heap tab, DDMS, 242
Hebrew language support, 10
height attribute, 33, 83, 101
hide() and show() method, 360
Hindu language support, 10
hint attribute, 83
Honeycomb code name, platform and API levels, 55
horizontal value, 101, 105-106, 125
HorizontalScrollView controls, 186
horizontalSpacing attribute, 229
HttpTransportSE object, 606
HttpURLConnection class, 598-602

I

Ice Cream Sandwich code name, platform and API levels, 55
icon attribute, 62, 361, 634
Icon Menus, 326
id attribute, 41
ifRoom value, 362
ImageAdapter (custom), 234-235
ImageView controls
  animations
    collecting and sequencing, 466-470
    frame-by-frame, 446-447, 449-453
    loading, 460-462
    tweening, 454-455
  attributes, 125
    src, 171-172
  description, 124
  FrameLayout, 125-128
  ImageView object, 173
  setImageResource() method, 173
  ToggleButton control, image switching, 181-183
in (inches) unit of measurement, 42, 83, 154
IncomingHandler object, 616
inflate() method, 400
info.txt in TextView control, 205-206
initLoader() method, 583
InputStream class, 206, 601
inputType attribute, 84
insert() method, 386, 573, 578-580
insertOrThrow() method, 390
insertRec() method, 400-401
insert_rows() method, 398
instantiateItem() method, 239
integer arrays, 149-170
Intent class, 76, 518, 618
  email, extras, 546
  methods, 518
    createChooser(), 546
    putExtra(), 518
    sendBroadcast(), 518, 521
    setAction(), 518
    setType(), 546
    startActivity(), 546
<intent- filter> tags, 63
intents, 76-77
  alarms, 624-625
  broadcast intents
    receiving, 519-523
    sending, 518
  defining in AndroidManifest.xml file, 60-61
  email, 546-552
  explicit, 77, 81
  implicit, 77
  <intent- filter> tags, 63
Internet access/data retrieval
  HttpURLConnection class, 598-602
    InputStream object, 601
    openHttpURLConnection() method, 601
  SOAP Web Services, 602-607
    HttpTransportSE object, 606
    kSOAP library, 603, 606
    PropertyInfo object, 606
isChecked() method, 340
isLocationDisplayed() method, 500
isRouteDisplayed() method, 489, 500
isViewFromObject() method, 239

J

Java files, 30
Java perspective, DDMS, 48
JDK (Java Development Kit), 12-13

How can we make this index more useful? Email us at indexes@samspublishing.com

Custom Setup dialog, 13
Development Tools, 13
downloading, 12
installing Standard Edition, 13
Java Setup Wizard, 13
JRE (Java runtime environment), 13-14
Public JRE (Java runtime environment), 13
setting path, 20
Source Code, 13
version detected, 14
Windows, Linux, or Mac platforms, 13
Jelly Bean code name
platform and API levels, 55
SDK
antipiracy support, 11
Camera app, 10
communication improvements, 10
Face Unlock, 10
Google Now, 10
Google Play widgets, 11
Google Search, 11
Google Voice search, 10
Home screen, auto-arranging, 10
languages
supporting bidirectional text, 10
supporting new, 10
notifications
blocking, 10
expanding/collapsing, 10
text, pictures, and lists, 10
predictive keyboard, 10
Project Butter, 10
speech recognition, 10
visually impaired help, Gesture Mode combined with voice, 10
JPG files, 125, 170
JRE (Java runtime environment), 13-14

K

key attribute, 484
Keypad device driver, 11
KML (Keyhole Markup Language), 495-496
kSOAP library, 603, 606

L

label attribute, 62, 634
landscape mode, 138, 140-144
description, 138
fragments, 282, 288, 293
languages supported, 10
layout folder, 58, 147-148
Layout Params, 33
layout subdirectory, 58
layouts/containers
attributes
above, 113
alignBaseline, 114
alignBottom, 114
alignLeft, 114
alignParentBottom, 113
alignParentLeft, 113
alignParentRight, 113
alignParentTop, 113
alignRight, 114
alignTop, 113
below, 113
centerHorizontal, 33, 113
centerInParent, 113
centerVertical, 33, 40, 113
column, 134
columnSpan, 134
gravity, 101, 109-110, 176-177

height, 33, 83, 101, 132, 136
margin, 114
marginBottom, 114
marginLeft, 114
marginRight, 114
marginTop, 114
row, 134
rowSpan, 134
span, 131
toLeftOf, 113
toRightOf, 113
weight, 101, 107, 111
width, 33, 82, 101
content alignment, 43
creating, 78-79
dimensions, units of measurement, 42-43, 83, 123
Views and ViewGroups, 30, 33, 101
LBS (Location-Based Services), 490-494
left value, 43, 103
Libs subdirectory, 58, 149
LINEAR_ACCELERATION sensor type, 626
LinearGradient() constructor, 436-438
LinearInterpolator, 471
LinearLayout, 30
arrangements, 104-111
attributes, 101
description, 66, 102
lines, drawing, 427
lines attribute, 83
Linux kernel, 11
ListActivity class, 211, 217-219
ListAdapter, 211, 214
ListFragment, 301-305. *See also* DialogFragment; FragmentManager; fragments; PreferenceFragment
ListView control, built-in, 301-305
onCreateView() method, 302
ListPreference View, 313-314, 316-318
list_rows() method, 398
ListView controls, 186
applying Context Menus to, 354-357
attributes
choiceMode, values, 209
drawSelectorOnTop, 209, 212
entries, 209, 212
multipleChoiceModal, 209
transcriptMode, values, 209
creating, 211
data entry forms, rows, 410
definition, 67, 209
fragments, 283-293
ListActivity class, 211, 217-219
methods
onListItemClick(), 219
setListAdapter(), 219
populating through Adapter, 211, 214
ArrayAdapter, 214-217
CursorAdapter, 214
ListAdapter, 214
populating through string resources, 211-213
setOnItemClickListener() method, 213
LoaderManager class, 581-583
Location-Based Services, 490-494
LocationListener() method, 493
LogCat, 245, 248, 255
log messages, 255-256
methods, 255
logo attribute, 361

How can we make this index more useful? Email us at indexes@samspublishing.com

M

MAGNETIC_FIELD sensor type, 626

Manifest Editor, 64

Manifest file. *See* Android Manifest file

<manifest> tags, attributes, 62

map locations

 AVD-based applications, 485-486

 displaying, 496-499

  map markers, 507-515

  satellite view, 499

  streets and places, 497, 499

  traffic view, 499, 501

 Google Maps API, 499-501

 Google Maps-based applications, 486-489

 GPX/KML format, 496

 latitude and longitude values, 494, 497

  sending manually, 495

  through DDMS, 494

  translating with street addresses, geocoding, 502-506

 LBS (Location-Based Services), 490-494

MapActivity class, 488

 isRouteDisplayed() method, 489, 500

MapView controls

 attributes

  clickable, 488

  enabled, 488

 displaying locations, 497-499

  transparent overlays, 507-510

 methods

  getCenter(), 515

  setZoom(), 500, 515

margin attribute, 114

marginBottom attribute, 114

marginLeft attribute, 114

marginRight attribute, 114

marginTop attribute, 114

match_parent value, 33, 82-83, 103, 110

 TableLayout, 132

maxHeight attribute, 83

maxSdkVersion="15" attribute, <uses-sdk> tags, 62

maxWidth attribute, 83

MD5 fingerprint, 483-484

Media Store content provider, 559

MediaController controls, 198

MediaPlayer

 Button control, attributes, 189

 methods

  create(), 189, 191, 203

  pause(), 191, 204

  start(), 189, 191

 ProgressBar control, 203-204

Memory device driver, 11

Menu Items, 358-359

 onOptionsItemSelected() method, 361-362

menu subdirectory, 148-149

menus

 Context Menus

  applying to ListView control, 354-357

  creating (coding), 349-353

  creating (XML), 336-344

  definition (XML), 324

 Menu button, deprecation of, 358

 Options Menus

  in ActionBar, 359

  creating (coding), 349-353

  creating (XML), 325-332

  defining (coding), 345-346

  Expanded Menus (XML), 324

  Icon Menus (XML), 323-324

 Submenus

  creating (coding), 346-353

  creating (XML), 332-336

  definition (XML), 324

message constant, 41

minHeight attribute, 83

minSdkVersion attribute, 62, 377, 635

minSdkVersion="15" attribute, 62

minWidth attribute, 83

mm (millimeters) unit of measurement, 42, 83, 154

monospace typeface, 444

moveToFirst() method, 392

moveToNext() method, 392

moveToPosition() method, 392

moveToPrevious() method, 392

multipleChoice value, 209

multipleChoiceModal attribute, 209

N

name attribute, 606

Name option, Threads tab, DDMS, 242

namespaces, 33

Near Field Communication, 10

NegativeButton() method, 309-310

Network Statistics tab, DDMS, 243

never value, 362

newInstance() method, 308

none value, 83, 125, 209, 229

normal value, 209

Notification class, 524, 529

  public members, 524-525

  setLatestEventInfo() method, 525

NotificationBuilder class, 525-527

  methods

    setAutoCancel(), 525

    setContentIntent(), 525

    setContentText(), 525

    setSmallIcon(), 525

    setTicker(), 525

    setWhen(), 525

NotificationManager class, 527-530

  methods

    getSystemService() method, 527

    notify() method, 527

notifications

  blocking, 10

  expanding/collapsing, 10

  text, pictures, and lists, 10

  via status bars, 523

notify() method, 527

number value, 84

numColumns attribute, 228-229

numericShortcut attribute, 375

O

On and Off values, 174-179

onAccuracyChanged() method, 627

onActivityCreated() method, 283

onAttach method, 283

onBind() method, 607-608, 614

onCallStateChanged() method, 554, 556

onCancelled() method, 595

onCheckedChanged() method, 87

onClick() method, 68, 87-88, 128, 176, 189, 408

onClickListener interface, 68, 128, 176, 182

onContextItemSelected() method, 340

onCreate() method, 34, 60, 283

onCreateContextMenu() method, 338-340, 353

onCreateDialog() method, 260-261, 309

onCreateLoader() method, 583

onCreateOptionsMenu() method, 35, 328, 398

onCreateView() method, 283, 286, 307

onDateSet() method, 270, 278
onDestroy() method, 60, 283, 608
onDestroyView() method, 283
onDetach() method, 283
onDraw() method, 423
oneshot attribute, 448
onItemClick() method, 213, 217, 230
onItemSelected() method, 223
onKey() method, 86, 476
OnKeyListener, 86
onListItemClick() method, 219
onLoaderReset() method, 583
onLoadFinished() method, 583
onLocationChanged(), 493, 500
onNothingSelected() method, 223
onOptionsItemSelected() method, 328, 353, 361-362, 366
onPageSelected() method, 236, 239
onPause() method, 60, 283, 627
onPostExecute() method, 595, 601
onPreExecute() method, 595, 597
onPreferenceClick() method, 315
onPrepareDialog() method, 260
onProgressUpdate() method, 594-595, 597
onProviderDisabled(), 493
onProviderEnabled(), 493
onReceive() method, 519, 521
onResume() method, 60, 283, 627
onSensorChanged() method, 627
onServiceConnected() method, 616, 618
onServiceDisconnected() method, 616, 618
onStart() method, 60, 283
onStartCommand() method, 607-608
onStatusChanged(), 494
onStop() method, 60, 283
onTabSelected() method, 379
onTabUnselected() method, 379
onTimeSet() method, 274, 278
onUpgrade() method, 390
open() method, 206
OpenGL support, Surface Manager, 11
openHttpURLConnection() method, 601
openReadable() method, 390, 393
Options Menus
  in ActionBar, 359
  attributes, alphabeticShortcut and numericShortcut, 331
  creating by coding, 349-353
    add() method, parameters, 345
    defining, 345-346
    onCreateOptionsMenu() method, 346, 353
    onOptionsItemSelected() method, 353
    setIcon() method, 346
  creating in XML, 325-332
    checkable and checked attributes, check boxes, 331
    Expanded Menus, 324
    Icon Menus, 323-324
    Icon Menus, title and icon attributes, 326
    onCreateOptionsMenu() method, 328, 398-400
    onOptionsItemSelected() method, 328
orientation attribute, 101, 105-106
Outline pane, Debug perspective, 248
ovals, 433-434
Overflow Menu, 359, 362
overlays (maps), 507-515
OvershootInterpolator, 471

P

package attribute, <manifest> tags, 62

Package Explorer

- Android Manifest Editor, 64
- Android XML File, 78
- project tree, 56

padding attribute, 101, 103, 114

paddingBottom attribute, 114

paddingLeft attribute, 114

paddingRight attribute, 114

paddingTop attribute, 114

PagerAdapter, 235-236, 239

Paint and Canvas objects, 421-423

- colors, 423-424
- displaying text, 443-444
- drawing
  - arcs, 434-435
  - bitmaps, 441-443
  - canvas height and width, 424-425
  - circles, 424
  - defining drawing paths, 429-430
  - lines, 427
  - ovals, 433-434
  - points, 425
  - rectangles, 430-431
  - rectangles, rounded, 432-433
- gradients, 436
  - LinearGradient, 436-438
  - RadialGradient, 439
  - SweepGradient, 440
- paint antialiasing, 425
- paint styles, 425
- strokes
  - cap, 426
  - width, 426

password attribute, 83

paths, drawing, 429-430

pause() method, 191, 204

PendingIntent class, 524-525, 529, 554

- alarms, 619-620, 624-625
- getActivity() method, 525

phone value, 84

PhoneStateListener class, 554-556

- onCallStateChanged() method, 554, 556

pivotX and pivotY attributes, 456-457

Places card, 10

Play Store. *See* Google Play Store

PNG files, 125, 170-171

points, 425

populate() method, 514

portrait mode, 140-142, 144, 282

- description, 138
- fragments, 282, 288

PositiveButton() method, 309-310

post() method, 592

postInvalidate() method, 510

predictive keyboard, 10

Preference View, 312-313

PreferenceCategory View, 312

PreferenceFragment, 311-318. *See also* DialogFragment; FragmentManager; fragments; ListFragment

- methods
  - addPreferencesFromResource(), 315
  - onPreferenceClick(), 315
- Views, 312-313. *See also* specific views

Preferences, SDK installation, 24

PreferenceScreen View, 312

PRESSURE sensor type, 626

ProgressBar controls

- definition, 199

max attribute, 200
methods
postDelayed(), 204
setMax(), 200
setProgress(), 200
updateProgressBar(), 203-204
styles, 200
proguard.cfg file, 58
Project Butter, 10
project.properties file, 58
prompt attribute, 222
PropertyInfo object, 606
<provider> tags, 63
PROXIMITY sensor type, 626
pts (points) unit of measurement, 42, 83
dimen resources, 154
publishProgress() method, 594, 597
put() method, 390
putExtra() method, 518
px (pixels) unit of measurement
controls, 42-43
EditText, 83
dimen resources, 154

Q–R

query() method, 386, 390-392, 572

R class, 150
RadialGradient() constructor, 439
RadioButton controls
checked/unchecked states, 87, 91
definition, 67
methods
check(), 92
getCheckedRadioButtonId(), 92
isChecked(), 91
onClick(), 94, 97
toggle(), 91
Submenus, 347-348
RadioGroup element, 91, 93-95
raw subdirectory, 149
<receiver> tags, 63
rectangles, 430-431
rounded, 432-433
/referenced libraries folder, 58
registerForContextMenu() method, 338-339
registerListener() method, 627
RELATIVE_HUMIDITY sensor type, 626
RelativeLayout, 30, 33
arrangements, 115-117, 119-120
control attributes
alignment, in relation to other controls, 113-114
location, relative to container, 113
position, in relation to other controls, 113
spacing, between control and container, 114
spacing, for containers and views, 114
description, 66, 102
screen orientation, 138-139
text, adding, 39-40
reload() method, 478
remove() method, 295
removeDialog() method, 260
repeatCount attribute, 463
repeatMode attribute, 463
replace() method, 295
Fragment1Activity, 295

requestLocationUpdates(), 493

res folder/resources, 57-58, 147

- anim subdirectory, 149
- *versus* assets directory, 204
- color subdirectory, 149
- drawable resources, 148
  - formats supported, 170
  - -hdpi subdirectory, 58, 147-149
  - image additions, 177-178, 236, 370
  - image resolutions, 170, 183, 236
  - -ldpi subdirectory, 58, 147-149
  - -mdpi subdirectory, 58, 147-149
  - referencing files, 171
  - -xhdpi subdirectory, 58, 148-149
- layout folder, 58, 147-148
- Libs subdirectory, 149
- menu subdirectory, 149
- naming conventions, 150
- R class, 150
- raw subdirectory, 149
  - audio files, 187
- values folder, 58, 147-149
  - arrays.xml, 149, 165-170
  - color.xml, 149, 156-158
  - dimens.xml, 149, 153-155
  - strings.xml, 149, 150-153
  - styles.xml, 149, 159-162
- values-11 file, 149
- values-14 file, 149
- XML subdirectory, 149

resizeMode attribute, 125

Restart attribute, 463

retrieveRows() method, 390, 393, 412

Reverse attribute, 463

reverse geocoding, 502-503

right value, 43, 103, 110-111

RingtonePreference View, 312, 316-318

rotate animations, 453, 456-457

RotateAnimation class, 464, 466

ROTATION_VECTOR sensor type, 626

row attribute, 134

rows

- GridLayout, 134-135
- TableRow object, 129, 132

rowSpan attribute, 134

RTC alarm, 619

RTC_WAKEUP alarm, 619

run() method, 594

S

sans serif typeface, 444

scale animations, 453, 457-458

ScaleAnimation class, 464, 466

scaleType attribute, 125

.schema ADB command, 396

Screen Capture/options, Devices tab, DDMS, 241-242

screen orientation

- anchoring controls, 138-140
- description, 138
- layout definitions, 138, 140-144

scrollHorizontally attribute, 82-83

ScrollLayout layout, 30

ScrollView controls, 183

- fillViewPort attribute, 186
- ImageView control, 183-186

SD cards, video files

- loading, 195-197
- referencing, 198

SDK (Software Development Kit)

- ADT (Android Development Tools) plug-in
  - Android DDMS, 22, 48
  - Android Development Tools, 22

How can we make this index more useful? Email us at indexes@samspublishing.com

Android Hierarchy Viewer, 22
Android Traceview, 22
attaching ADT to Eclipse IDE, 24
downloading, 12
installing, 22
license agreement, 23
software updates, 24
uses, 19-20
Android Platform SDK Starter Package, downloading, 12
dialog window types, 259
Eclipse IDE (integrated development environment)
downloading, 12
installing, 20
uses, 19
Welcome screen, 20
Workbench, 21
workspace location, 20
JDK (Java Development Kit), 12-13
Custom Setup dialog, 13
Development Tools, 13
downloading, 12
installing Standard Edition, 13
Java Setup Wizard, 13
JRE (Java runtime environment), 13-14
Public JRE, 13
setting path, 20
Source Code, 13
version detected, 14
Windows, Linux, or Mac platforms, 13
menu types, 323-324
Preferences, SDK installation, 24
SDK jar file, 57
SDK Manager, 12, 16
ADB Restart window, 18
Google API, 484
installing packages, 16-18
Log window, 18
SDK Tools
installing, 14-16
user selection, 14
SDK Tools Setup Wizard, 14-15
Selection Widget, adapters, 214, 216
sendBroadcast() method, 518, 521
sendEmptyMessage() method, 592
sendMessage() method, 592
sendTextMessage() method, 537-538
sensor types
ACCELEROMETER, 626, 628-630
AMBIENT_TEMPERATURE, 626
data rates, 628
GRAVITY, 626
GYROSCOPE, 626
LINEAR_ACCELERATION, 626
MAGNETIC_FIELD, 626
methods
getDefaultSensor(), 627
getSensorList(), 627
getSystemService(), 626
onAccuracyChanged(), 627
onPause(), 627
onResume(), 627
onSensorChanged(), 627
registerListener(), 627
unregisterListener(), 627
PRESSURE, 626
PROXIMITY, 626
RELATIVE_HUMIDITY, 626
ROTATION_VECTOR, 626
SensorManager, 626
SensorManager, 626
sentences value, 83
serif typeface, 444

ServiceConnection interface, 616, 618

services. *See also* bound services; started services

<service> tags, 63

set() method, 619, 625

setAction() method, 518

setAdapter() method, 227, 230

setArguments() method, 296-297

setAutoCancel() method, 525

setBackgroundResource() method, 45

setBounds() method, 515

setBuiltInZoomControls() method, 499

setCenter() method, 500

setCheckable() method, 347

setChecked() method, 87, 347-348

setColumnCollapsed() method, 130

setContent() method, 34

setContentDescription() method, 378

setContentIntent() method, 525

setContentText() method, 525

setContentView() method, 423

setCurrentItem() method, 236

setDisplayShowHomeEnabled() method, 361

SetDisplayShowTitleEnabled() method, 379

setDisplayShowTitleEnabled() method, 361

setEllipsize() method, 45

setGravity() method, 43

setGroupCheckable() method, 347-348

setHeaderIcon() method, 340

setHeaderTitle() method, 340

setHeight() method, 45

setHomeButtonEnabled() method, 361, 366

setIcon() method, 261, 346, 378

setImageResource() method, 173

setInexactRepeating() method, 620

setLatestEventInfo() method, 525

setListAdapter() method, 219, 304

setListNavigationCallbacks() method, 380-381

setMediaController() method, 198

setMessage method, 261

setNavigationMode() method, 377, 380

setNegativeButton method, 261

setNeutralButton method, 261

setOnClickListener() method, 128

setOneShot() method, 453

setOnItemClickListener() method, 213

setOnItemSelectedListener() method, 223

setOrientation() method, 101

setPositiveButton method, 261

setRepeatCount() method, 463-464

setRepeating() method, 620, 625

setSatellite() method, 499, 501

setSmallIcon() method, 525

setStrokeCap() method, 426

setStrokeWidth() method, 426

setStyle() method, 425

setStyles() method, 425

setText() method, 45, 378

setTextColor() method, 45, 158

setTextSize() method, 45, 155, 443-444

setTextStyle() method, 45, 155

setThreshold() method, 227

setTicker() method, 525

setTimeInMillis method, 621

Settings content provider, 559

setTraffic() method, 499, 501

setTransformationMethod() method, 45

setType() method, 546

setTypeface() method, 45, 444-445

setVideoPath() method, 198

setWebClient() method, 481

setWhen() method, 525

setWidth() method, 45

setZoom() method, 515

SGL support, Surface Manager, 11

shortcuts

  Options Menus, 330-331

    alphabeticShortcut and numericShortcut attributes, 331

  SDK Tools, 15

  Submenu methods, 348

    setAlphabeticShortcut(), 348

    setNumericShortcut(), 348

    setShortcut() method, 348

shouldOverrideUrlLoading() method, 480, 482

show() method, 310, 360

showAsAction attribute, 362, 371, 375

showDialog() method, 260-261

showRec() method, 400, 414

shrinkColumns attribute, 130

Simple Secure Pairing, 10

SimpleOnPageChangeListener, 236

singleChoice value, 209

singleLine attribute, 45, 83

SMS Messages

  receiving, 541-545

    registering Broadcast Receiver class, 543

    seeking permissions, 543

    SMS PDUs (Protocol Data Unit), 542

  sending, 531-538

  permissions, 534

  status monitoring, 538

SmsManager class methods

  getDefault(), 537

  sendTextMessage(), 537-538

SmsMessage class methods

  createFromPdu(), 542

  getMessageBody(), 543

  getOriginatingAddress(), 543

  getTimestampMillis(), 543

SOAP Web Services, 602-607

  HttpTransportSE object, 606

  kSOAP library, 603, 606

  PropertyInfo object, 606

sp (scale independent pixels) unit of measurement, 42, 83

  dimen resources, 154

Space view, 134-135

SpacingWidth value, 229

span attribute, 131

speech recognition, 10

Spinner controls

  attributes

    entries, 222-223

    prompt, 222

  definition, 67, 220

  methods

    onItemSelected(), 223

    setOnItemSelectedListener(), 223

  populating through ArrayAdapter, 223-225

  populating through resources

    string arrays, 221-222

    strings, 220

  prompt attribute, 222

Sports card, 10

SQL DELETE ADB command, 397

SQL SELECT ADB command, 396-397

SQL UPDATE ADB command, 397

SQLHelper class, query() method, 390

SQLite relational databases

  commands with ADB

    .exit, 397

    .schema, 396

    SQL DELETE, 397

    SQL SELECT, 396-397

    SQL UPDATE, 397

    .tables, 396

Cursor class methods
getColumnIndexOrThrow(), 392
getColumnName(), 392
getColumnNames(), 392
getCount(), 392
getPosition(), 392
moveToFirst(), 392
moveToNext(), 392
moveToPosition(), 392
moveToPrevious(), 392
DatabaseManager class, 386-390, 392
addRow() method, 393
openReadable() method, 390, 393
retrieveRows() method, 390, 393
SQLHelper class, 390
SQLite library, 11
SQLiteDatabase class methods
insert(), 386
query(), 386, 390-392
SQLiteOpenHelper class, 385-387
addRow() method, 390
close() method, 385, 390
from DatabaseManager class, 390
execSQL() method, 390
getReadable() method, 390
getReadableDatabase() method, 385, 390
getWritableDatabase() method, 385, 390
insertOrThrow() method, 390
onCreate() method, 390
onUpgrade() method, 390
put() method, 390
src attribute, 125, 171-172
/src folder, 57
/src/com.androidunleashed.welcomemsg package name, 57
start() method, 189, 191
AnimationDrawable class, 449
Start Method Profiling, Devices tab, DDMS, 241
startActivity() method, 77-78, 291, 546
started services, 609-611
definition, 607
interacting with, 611-614
methods
bindService(), 608
onBind(), 607-608
onDestroy(), 608
onStartCommand(), 607-608
startService(), 608
stopService(), 608
unbindService(), 608
Service class, 607
startService() method, 608
Status option, Threads tab, DDMS, 242
STD (subscriber trunk dialing) data
displaying, 584-585
entering, 566-568, 580, 588-589
retrieving, 569-570, 587
storing, 568
stime option, Threads tab, DDMS, 242
stop() method, 449
Stop Process, Devices tab, DDMS, 241
stopService() method, 608
stretchColumns attribute, TableLayout, 129-130
stretchMode attribute, 229
string resources, 149-151. *See also* array resources
getString() method, 152-151
name properties/IDs, 151
populating
ListView control, 211-213
Spinner control, 220-221
tags for bold, italics, and underline, 152-153

strokes

- cap, 426
- width, 426

style resources, 149, 159-160

- name property/IDs, 159-192

submenus

- creating by coding, 346-353
  - addSubMenu() method, 346
  - check boxes, 347-348, 375
  - radio buttons, 347-348
  - setCheckable() method, 347
  - setChecked() method, 347-348
  - setGroupCheckable() method, 347-348
- creating in XML, 332-336
  - checkableBehavior attribute, <group> node, 334
- definition, 324

Surface Manager, 11

SweepGradient() constructor, 440

T

tabbed ActionBar, 377-380

- methods
  - addTab(), 377
  - getActionBar(), 379
  - onTabSelected(), 379
  - onTabUnselected(), 379
  - setContentDescription(), 378
  - SetDisplayShowTitleEnabled(), 379
  - setIcon(), 378
  - setNavigationMode(), 377
  - setText(), 378
- TabListener, 378-379

TableLayout, 30

- arrangements, 131-133
- columns, operations on, 129-131
- controls, arranging, 132
- description, 67, 102, 129

TableRow object, 129, 132

.tables ADB command, 396

TabListener, 378-379

targetSdkVersion attribute, <uses-sdk> tags, 62

telephony services, 553

- outgoing calls, 553-556
- permissions, 556-557
- phone state changes, 554-556

text

- displaying, 443-444
- drawing, 443-444
- sizes, 443-444
- typefaces, 444-445

text attribute, 40, 45

text value, 84

textAutoCorrect value, 84

textCapCharacters value, 84

textCapWords value, 84

textColor attribute, 44-46

textEmailAddress value, 84

textMultiLine value, 84

textOn and textOff attributes, ToggleButton control, 174-179

textPassword value, 84

textSize attribute, 44-46, 83

textStyle attribute, 44-46

TextView controls, 30, 33, 66

- AbsoluteLayout arrangements, 123
- arrangements
  - FrameLayout, 126-128
  - GridLayout, 135-137
  - LinearLayout, 103
  - RelativeLayout, 117-121
  - TableLayout, 131-133

assigning to text

in activity file, 40-42

in layout file, 39-41

attributes, 33

gravity, 44-46, 190

gravity, values, 43

list of, 43-46

definition, 67

fragments, 283-293

instructions

to display text, 39-41

to play audio, 187-194

methods, onNothingSelected(), 223

Thai language support, 10

theme attribute, <activity> or <application> elements, 162-164

Thread ID option, Threads tab, DDMS, 242

threads. *See* background threads

Threads tab, DDMS, 242

Name, 242

Status, 242

stime, 242

Thread ID, 242

utime, 242

3D graphics, Surface Manager, 11

time value, 84

TimePickerDialog, 260, 271-272

Boolean values, 274

Calendar instance, 274

onTimeSet() method, 274, 278

tasks performed, 272-273

title attribute, 326

toAlpha attribute, 455

Toast class

constants

LENGTH_LONG, 75

LENGTH_SHORT, 75

displaying output, 75

methods, makeText(), 75

toDegrees attribute, 456

toggle() method, 87-91

ToggleButton controls

animations, frame-by-frame, 446-447, 450-453

attributes

background, 176-179

gravity, 176

layout_gravity, 176-177

textOn and textOff, 174-179

audio, playing, 190-194

images, switching, 181-183

toLeftOf attribute, 113

top value, 43, 103

toPixel() method, 511

toRightOf attribute, 113

toXDelta attribute, 458

toXScale attribute, 457

toYDelta attribute, 458

toYScale attribute, 457

transcriptMode, values, 209

Transit card, 10

translate animations, 453, 458-460

TranslateAnimation class, 464-465

true value, 83, 209, 488

tweening animations, 446

alpha animations, 453, 455-456

combining and sequencing, 457-458

defining with Java code, 464-466

defining with XML, 454-455

interpolators, 471-472

rotate animations, 453, 456-457

scale animations, 453, 457-458

translate animations, 453, 458-460

View animations, with XML, 454-455

How can we make this index more useful? Email us at indexes@samspublishing.com

2D graphics, Surface Manager, 11

typeface attribute, 45-46

U

UI (user interface)

- controls, 30
- creating in Java, HML, or combination, 64-65

unbindService() method, 608

units of measurement, dimensions, 42-43, 83, 123

unregisterListener() method, 627

Update Heap, Devices tab, DDMS, 240

update() method, 573-574

Update Threads, Devices tab, DDMS, 241

URIs (Universal Resource Identifiers), 560-561, 569-570

- authority, 560
- date path, 560
- id, 560
- standard prefixes, 560

<uses-configuration> tags, 636

<uses-feature> tags, 636

<uses-permissions> tags, 63-64, 636

<uses-sdk> tags, attributes, 62

utime option, Threads tab, DDMS, 242

V

values folder/directory, 58, 147-149

- arrays, 149, 165
  - integer arrays, 149-170
  - string arrays, 165-168, 221-222
- colors, 149, 156-158
  - Alpha channel, 156-157
  - hexadecimal RGB values, 156
- dimens, 149, 153-154
  - getDimension() method, 155
  - units of measurement, 154-155
- strings, 149
  - getString() method, 151-152
  - name properties/IDs, 151
  - tags for bold, italics, and underline, 152-153
- styles, 149
  - name property/IDs, 159-192

Variables pane, Debug perspective, 247, 254

versionCode/versionName attributes, <manifest> tags, 62, 633-634

versioning information

- attributes
  - icon, 634
  - label, 634
  - minSdkVersion, 635
  - versionCode, 633-634
  - versionName, 634
- tags
  - <uses-configuration>, 636
  - <uses-feature>, 636
  - <uses-permissions>, 636

vertical value, 101, 111

verticalSpacing attribute, 229

video files

- SD cards
  - loading video, 195-197
  - referencing video, 198
- SDCARD folder, 195

VideoView controls, 197-198

View animations
  frame-by-frame animations, 446
    defining with Java code, 451-453
    defining with XML, 446-451
  tweening animations, 446
    alpha animations, 453, 455-456
    combining and sequencing, 457-458
    defining with Java code, 464-466
    defining with XML, 454-455
    rotate animations, 453, 456-457
    scale animations, 453, 457-458
    translate animations, 453, 458-460
ViewGroups, 30
ViewPager controls, 236-239
  definition, 235
  image gallery, 236-237, 239
  methods
    destroyItem(), 239
    getCount(), 239
    instantiateItem(), 239
    isViewFromObject(), 239
    onPageSelected(), 236, 239
    setCurrentItem(), 236
  PagerAdapter, 235-236, 239
Views, 30, 33
  Fragment class, 282
  portrait and landscape modes, 289
visually impaired help, Gesture Mode combined with voice, 10

W

Weather card, 10
WebView controls, 473-477
  Internet access permissions, 477-480
  methods
    canGoBack(), 478
    canGoBackOrForward(), 477-478
    canGoForward(), 477-478
    clearCache(), 478
    clearHistory(), 478
    goBack(), 477-478
    goBackOrForward(), 478
    goForward(), 477-478
    reload(), 478
  navigation, 477
WebViewClient class, 480-482
weight attribute, 101, 107, 111
width attribute, 33, 82, 101
Wi-Fi device driver, 11
withAppendedId() method, 573
words value, 83
wrap_content value, 45, 103, 132
wrap_parent value, 33

X

x and y coordinates, AbsoluteLayout, 121, 123
XML files, 30
  AndroidManifest, 80-81
  event handling, 72-75
  screen definitions, 31
XML subdirectory, 149

How can we make this index more useful? Email us at indexes@samspublishing.com