Introduction to Android Application Development, Android Essentials, Fifth Edition

Chapter 6

Managing Application Resources
Chapter 6
Overview

- Understand what resources are and how to use them in our applications
- Set simple resource values using Android Studio
- Work with different types of resources
- Work with and define Android application screen elements and layouts
- Reference system resources
What Are Resources?

- All Android applications are composed of two things:
  1. Functionality (code instructions)
     - Functionality is the code that determines how your application behaves.
     - This includes any algorithms that make the application run.
  2. Data (resources)
     - Resources include text strings, styles and themes, dimensions, images and icons, audio files, videos, and other data used by the application.
- Many of the code examples provided in this chapter are taken from the SimpleResourceView, ResourceRoundup, and ParisView applications.
  - The source code for these applications is provided for download on the book’s website (http://introductiontoandroid.blogspot.com).
Storing Application Resources

- Android resource files are stored separately from the `.java` class files in the Android project.
  - Most common resource types are stored in XML.
  - You can also store raw data files and graphics as resources.
  - Resources are organized in a strict directory hierarchy.
  - All resources must be stored under the `res/` project directory in specially named subdirectories that must be lowercase.
## Storing Application Resources (Cont’d)

<table>
<thead>
<tr>
<th>Resource Subdirectory</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>res/drawable/</td>
<td>Graphics resources</td>
</tr>
<tr>
<td>res/layout/</td>
<td>User interface resources</td>
</tr>
<tr>
<td>res/menu/</td>
<td>Menu resources for showing options or actions in your activities</td>
</tr>
<tr>
<td>res/mipmap/</td>
<td>App launcher icon resources</td>
</tr>
<tr>
<td>res/values/</td>
<td>Simple data such as strings, styles and themes, and dimensions</td>
</tr>
</tbody>
</table>
Storing Application Resources (Cont’d)

- Each resource type corresponds to a specific resource subdirectory name.
  - For example:
    - All graphics are stored under the res/drawable/ directory structure.
- Resources can be further organized in a variety of ways using even more specially named directory qualifiers.
  - For example:
    - res/drawable-hdpi/ stores graphics for high-density screens
    - res/drawable-ldpi/ stores graphics for low-density screens
    - res/drawable-mdpi/ stores graphics for medium-density screens
    - res/drawable-xhdpi/ stores graphics for extra-high-density screens
    - res/drawable-xxhdpi/ stores graphics for extra-extra-high-density screens
  - If you had a graphic resource that was shared by all screens, you would simply store that resource in the res/drawable/ directory.
Storing Application Resources (Cont’d)

- Android Studio makes adding resources simple.
  - It automatically detects new resources when you add them to the appropriate project resource subdirectory under `res/`.
  - These resources are compiled, resulting in the generation of the `R.java` source file.
- This file enables you to access your resources programmatically.
Resource Value Types

- Android applications rely on many different types of resources:
  - These include text strings, graphics, color schemes, and other resources for user interface design.

- These resources are stored in the `res` directory of your Android project in a strict (but reasonably flexible) set of directories and files.

- All resource filenames must be lowercase and simple.
  - They must use letters, numbers, and underscores only.
## Resource Value Types (Cont’d)

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property animations</td>
<td>res/animator/</td>
</tr>
<tr>
<td>Tweened animations</td>
<td>res/anim/</td>
</tr>
<tr>
<td>Color state lists</td>
<td>res/color/</td>
</tr>
<tr>
<td>Drawables</td>
<td>res/drawable/</td>
</tr>
<tr>
<td>Layouts</td>
<td>res/layout/</td>
</tr>
<tr>
<td>Menus</td>
<td>res/menu/</td>
</tr>
<tr>
<td>Arbitrary raw files</td>
<td>res/raw/</td>
</tr>
<tr>
<td>Simple values</td>
<td>res/values/</td>
</tr>
<tr>
<td>Arbitrary XML</td>
<td>res/xml/</td>
</tr>
</tbody>
</table>
Storing Primitive Resource Types

- Simple resource value types, such as strings, colors, dimensions, and other primitives, are stored under the `res/values/` project directory in XML files.
- Each resource file under the `res/values/` directory should begin with the following XML header:
  ```xml
  <?xml version="1.0" encoding="utf-8"?>
  ```
- Next comes the root node `<resources>` followed by the specific resource element types such as `<string>` or `<color>`.
- Each resource is defined using a different element name.
- Primitive resource types simply have a unique name and a value, like this color resource:
  ```xml
  <color name="myFavoriteShadeOfRed">#800000</color>
  ```
Tip

- Although the XML filenames are arbitrary, the best practice is to store your resources in separate files to reflect their types, such as strings.xml, colors.xml, and so on.

- However, there’s nothing stopping developers from creating multiple resource files for a given type, such as two separate XML files called bright_colors.xml and muted_colors.xml.
Storing Graphics and Files

- You can also store numerous other types of resources, such as graphics, arbitrary XML files, and raw files.
  - These types of resources are stored in specially named directories according to their type.
  - For example:
    - Graphics are stored in the res/drawable/ directory structure.
    - XML files can be stored in the res/xml/ directory.
    - Raw files can be stored in the res/raw/ directory.

- Make sure you name resource files appropriately because the resource name for graphics and files is derived from the filename of the specific resource.
  - For example:
    - A file called flag.png in the res/drawable/ directory is given the name R.drawable.flag.
Storing Other Resource Types

- All other resource types—be they tweened animation sequences, color state lists, or menus—are stored in special XML formats in various directories, as previously discussed.

- Each resource must be uniquely named.
Understanding How Resources Are Resolved

- The Android platform has a very robust mechanism for loading the appropriate resources at runtime.
  - You can organize Android project resources based on more than a dozen different criteria.
  - Resources stored at the directory hierarchy are considered default resources.
  - Under certain conditions, you can also supply special versions of your resources to load instead of the defaults.
- These specialized resources are called alternative resources.
- Some common reasons that developers use alternative resources:
  - Internationalization
  - Localization
  - Accommodating different device screens and orientations
Understanding How Resources Are Resolved (Cont’d)

- Default and alternative resources are best illustrated by example.
  - Let’s presume that we have a simple application with string, graphics, and layout resources.
  - In this application, the resources are stored in the top-level resource directories.
    - For example:
      - res/values/strings.xml
      - res/drawable/mylogo.png
      - res/layout/main.xml
  - No matter what Android device you run this application on, the same resource data is loaded and used.
  - This application uses only default resources.

- But what if we want our application to use different graphics sizes based on the screen density?
  - We could use alternative resources to do this.
Understanding How Resources Are Resolved (Cont’d)

- Low-density screens
  - res/drawable-ldpi/mylogo.png
- Medium-density screens
  - res/drawable-mdpi/mylogo.png
- High-density screens
  - res/drawable-hdpi/mylogo.png
- Extra-high-density screens
  - res/drawable-xhdpi/mylogo.png
- Extra-extra-high-density screens
  - res/drawable-xxhdpi/mylogo.png
- Layout loaded in portrait mode
  - res/layout-port/main.xml
- Layout loaded in landscape mode
  - res/layout-land/main.xml
Accessing Resources Programmatically

- Developers access specific application resources using the `R.java` class file and its subclasses.
- These are automatically generated when you add resources to your project if you use Android Studio.
- You can refer to any resource identifier in your project by its name, which is why it must be unique.
  - For example:
    - A string resource named `strHello` defined within the resource file called `res/values/strings.xml` is accessed in the code as follows:
      - `R.string.strHello`
To access resources programmatically:
- First, retrieve the resource instance for your application Context (android.content.Context).
  - In this case, the Activity class extends Context.
- Then use the resource instance to get the appropriate kind of resource you want.
  - The Resources class android.content.res.Resources has helper methods for handling every kind of resource.

A simple way to retrieve the string text is to call the getString() method of the Resources class, like this:

```java
String myString = getResources().getString(R.string.strHello);
```
Adding Simple Resource Values in Android Studio

- To illustrate how to add resources in Android Studio, let’s look at an example.
- Create a new Android project and navigate to the `res/values/strings.xml` file in the Android Studio and double-click the file to edit it.
- Alternatively, you can use the Android project included with the book called ResourceRoundup to follow along.
Adding Simple Resource Values in Android Studio (Cont’d)

```xml
<resources>

<!-- Some basic strings. -->
<string name="app_name">Resource Roundup!</string>
<string name="hello">Hi!</string>

<string name="action_settings">Settings</string>
<!-- This string has some bold styling. -->
<string name="boldhello"><b>Hello</b></string>

<!-- This is a simple format string. -->
<string name="simpleformatString">Your score is $1$d out of $2$d! You $3$s.</string>
<!-- This is a format string with some styling. -->
<string name="formatStringWithTwoNumbersAndAString">Your score is $1$d out of $2$d! You &amp;lt;i&gt;$3$s&amp;lt;/i&gt;.</string>

<!-- These are some strings used in arrays.xml. -->
<string name="imageView02">ImageView2</string>
<string name="imageView01">ImageView1</string>
<string name="greenRectangle">Green Rectangle</string>
<string name="start">Start!</string>
<string name="stop">Stop!</string>
<string name="vroomAccelerate">Vroom! Accelerate!</string>
</resources>
```
Now add some resources to the XML; specifically, create the following resources:

- A color resource named `prettyTextColor` with a value of `#ff0000`
- A dimension resource named `textPointSize` with a value of `14pt`
- A drawable resource named `redDrawable` with a value of `#F00`
Adding Simple Resource Values in Android Studio (Cont’d)

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ResourceRoundup</string>
    <string name="hello">Hello World, ResourceRoundupActivity</string>
    <color name="prettyTextColor">#ff0000</color>
    <dimen name="textPointSize">14pt</dimen>
    <drawable name="redDrawable">#F00</drawable>
</resources>
Adding Simple Resource Values in Android Studio (Cont’d)

- Save the `strings.xml` resource file.
- Android Studio automatically generates the `R.java` file in your project, with the appropriate resource IDs, which enables you to programmatically access your resources after they are compiled into the project.
- You may navigate to your `R.java` file by switching to the `Active Tool Window of the Project tab` of Android Studio to `Project Files`.
- Then, expand the `app/` folder so that the `build/` folder is visible.
- Then, expand further into `r/debug`, and finally expand into `com/introtoandroid/resourceroundup`, and you should see the `R.java` file shown on the following slide.
Adding Simple Resource Values in Android Studio (Cont’d)
**Adding Simple Resource Values in Android Studio (Cont’d)**

```java
class R {
    public static final class attr {
    }
    public static final class color {
        public static final int prettyTextColor=0x7f050000;
    }
    public static final class dimen {
        public static final int textPointSize=0x7f060000;
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
        public static final int redDrawable=0x7f020001;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int hello=0x7f040001;
    }
}
```
String myString =
getResources().getString(R.string.hello);
int myColor =
    ContextCompat.getColor(context,
R.color.prettyTextColor);
float myDimen =
getResources().getDimension(R.dimen.textPointSize);
ColorDrawable myDraw = (ColorDrawable) ContextCompat.
    getDrawable(R.drawable.redDrawable);
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Use Some Resources</string>
    <string name="hello">Hello World, UseSomeResources</string>
    <color name="prettyTextColor">#ff0000</color>
    <dimen name="textPointSize">14pt</dimen>
    <drawable name="redDrawable">#F00</drawable>
    <string-array name="flavors">
        <item>Vanilla</item>
        <item>Chocolate</item>
        <item>Strawberry</item>
    </string-array>
</resources>
String[] aFlavors =
    getResources().getStringArray(R.array.flavors);
Adding Simple Resource Values in Android Studio (Cont’d)

- You now have a general idea of how to add simple resources using the Android Studio editor, but there are quite a few different types of data available to add as resources.
- It is a common practice to store different types of resources in different files.
- For example, you might store the strings in `res/values/strings.xml`, but store the `prettyTextColor` color resource in `res/values/colors.xml` and the `textPointSize` dimension resource in `res/values/dimens.xml`.
- Reorganizing where you keep your resources in the resource directory hierarchy does not change the names of the resources or the code used earlier to access the resources programmatically.
Working with Different Types of Resources

- In this section, we look at the specific types of resources available for Android applications, how they are defined in the project files, and how you can access this resource data programmatically.
- For each type of resource, you learn what types of values can be stored and in what format.
Working with String Resources

- String resources are among the simplest resource types available.
- String resources might show text labels on form views and for help text.
- By default, the application name is stored as a string resource.
- String resources are defined in XML under the res/values/ project directory and compiled into the application package at build time.
- All strings with apostrophes or single straight quotes need to be escaped or wrapped in double straight quotes.
## Working with String Resources (Cont’d)

<table>
<thead>
<tr>
<th>String Resource Value</th>
<th>Displays As</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello, World</td>
<td>Hello, World</td>
</tr>
<tr>
<td>&quot;User's Full Name:&quot;</td>
<td>User’s Full Name:</td>
</tr>
<tr>
<td>User's Full Name:</td>
<td>User’s Full Name:</td>
</tr>
<tr>
<td>She said, &quot;Hi.&quot;</td>
<td>She said, “Hi.”</td>
</tr>
<tr>
<td>She's busy but she did say, &quot;Hi.&quot;</td>
<td>She’s busy but she did say, “Hi.”</td>
</tr>
</tbody>
</table>
You can edit the `strings.xml` file using the Resources tab, or you can edit the XML directly by clicking the file and choosing the `strings.xml` tab. After you save the file, the resource identifiers are automatically added to your `R.java` class file. String values are appropriately tagged with the `<string>` tag and represent a name/value pair. The `name` attribute is how you refer to the specific string programmatically, so name these resources wisely. Here's an example of the string resource file `res/values/strings.xml`:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Resource Viewer</string>
    <string name="test_string">Testing 1,2,3</string>
    <string name="test_string2">Testing 4,5,6</string>
</resources>
```
Bold, Italic, and Underlined Strings

- You can also add three HTML-style attributes to string resources.
- These are bold, italic, and underlining.
- You specify the styling using the `<b>`, `<i>`, and `<u>` tags, respectively.
- For example:

```xml
<string name="txt">
  <b>Bold</b>, <i>Italic</i>, <u>Line</u>
</string>
```
Using String Resources as Format Strings

- You can create format strings, but you need to escape all bold, italic, and underlining tags.
- For example, this text shows a score and the “win” or “lose” string:
  
  ```xml
  <string name="winLose">
    Score: %1$d of %2$d! You %3$s.
  </string>
  ```
  
  If you want to include bold, italic, or underlining in this format string, you need to escape the format tags.
  For example, if you want to italicize the “win” or “lose” string at the end:
  
  ```xml
  <string name="winLoseStyled">
    Score: %1$d of %2$d! You &lt;i&gt;%3$s&lt;/i&gt;.
  </string>
  ```
  
  Those of you familiar with XML will recognize this as standard XML escaping. After the standard set of XML escape characters is parsed, the string is interpreted with the formatting tags.
  
  - You also need to escape single quotes (‘ is &apos;), double quotes (" is &quot;), and ampersands (& is &amp;).
### Using String Resources Programmatically

- There are two primary ways in which you can access a string resource.
- The following code accesses your application’s string resource named `hello`, returning only the string:
  ```java
  String myStrHello =
  getResources().getString(R.string.hello);
  ```
- All HTML-style attributes (bold, italic, and underlining) are stripped from the string.
- You can also access the string and preserve the formatting by using this alternative method:
  ```java
  CharSequence myBoldStr =
  getResources().getText(R.string.boldhello);
  ```
Using String Resources Programmatically (Cont’d)

String mySimpleWinString;
mySimpleWinString =
    getResources().getString(R.string.winLose);
String escapedWin =
    TextUtils.htmlEncode(mySimpleWinString);
String resultText =
    String.format(mySimpleWinString, 5, 5, escapedWin);

- The resulting text in the resultText variable is:
  - Score: 5 of 5! You Won
Using String Resources Programmatically (Cont’d)

String myStyledWinString;
myStyledWinString =
    getResources().getString(R.string.winLoseStyled);
String escapedWin =
    TextUtils.htmlEncode(myStyledWinString);
String resultText =
    String.format(myStyledWinString, 5, 5, escapedWin);
CharSequence styledResults = Html.fromHtml(resultText);

- The resulting text in the styledResults variable is:
  - Score: 5 of 5! You <i>Won</i>
Working with Quantity Strings

- A special resource type called `<plurals>` can be used to define strings that are useful for changing a word’s grammatical quantity form.
- Here is an example string resource file with the resource path of `res/values/strings.xml` that defines two different quantity forms of a particular animal name that changes based on the context:

```xml
<resources>
  <plurals name="quantityOfGeese">
    <item quantity="one">You caught a goose!</item>
    <item quantity="other">You caught %d geese</item>
  </plurals>
</resources>
```
The singular form for this particular animal is *goose* and the plural form is *geese*. The `%d` value is used so we can display the exact quantity of geese to the user. To work with pluralized resources in your code, the method `getQuantityString()` can be used to retrieve a plural string resource as shown here:

```java
int quantity = getQuantityOfGeese();
Resources plurals = getResources();
String geeseFound = plurals.getQuantityString(
    R.plurals.quantityOfGeese, quantity,
    quantity);
```
### Working with Quantity Strings (Cont’d)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>Used for languages that have words with a zero quantity form.</td>
</tr>
<tr>
<td>one</td>
<td>Used for languages that have words with a singular quantity form.</td>
</tr>
<tr>
<td>two</td>
<td>Used for languages that have words for specifying two.</td>
</tr>
<tr>
<td>few</td>
<td>Used for languages that have words for specifying a small quantity batch.</td>
</tr>
<tr>
<td>many</td>
<td>Used for languages that have words for specifying a large quantity batch.</td>
</tr>
<tr>
<td>other</td>
<td>Used for languages that have words that do not have a quantity form.</td>
</tr>
</tbody>
</table>
Working with String Arrays

- You can specify lists of strings in resource files.
  - This is a good way to store menu options and drop-down list values.
  - String arrays are defined in XML under the res/values/project directory.
  - They are compiled into the application package at build time.

- String arrays are appropriately tagged with the `<string-array>` tag.
  - They usually include a number of `<item>` child tags, one for each string in the array.
Working with String Arrays (Cont’d)

<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="flavors">
    <item>Vanilla</item>
    <item>Chocolate</item>
    <item>Strawberry</item>
    <item>Coffee</item>
    <item>Sherbet</item>
  </string-array>
  <string-array name="soups">
    <item>Vegetable minestrone</item>
    <item>New England clam chowder</item>
    <item>Organic chicken noodle</item>
  </string-array>
</resources>
Working with String Arrays (Cont’d)

- Accessing string array resources is easy.
- The method `getStringArray()` retrieves a string array from a resource file, in this case one named `flavors`:

```java
String[] aFlavors = getResources().getStringArray(R.array.flavors);
```
Working with Boolean Resources

- Other primitive types are supported by the Android resource hierarchy.
- Boolean resources can be used to store information about application game preferences and default values.
  - These are defined in XML under the res/values/ project directory and compiled into the application package at build time.
Defining Boolean Resources in XML

- Boolean values are appropriately tagged with the `<bool>` tag and represent a name/value pair.
- The `name` attribute is how you refer to the specific Boolean value programmatically, so name these resources wisely.
- Here's an example of the Boolean resource file `res/values/bools.xml`:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="bOnePlusOneEqualsTwo">true</bool>
    <bool name="bAdvancedFeaturesEnabled">false</bool>
</resources>
```
Using Boolean Resources Programmatically

- To use a Boolean resource in code:
  - Load it using the `getBoolean()` method of the `Resources` class.

- To access your application’s Boolean resource named `bAdvancedFeaturesEnabled`:

```java
boolean bAdvancedMode = getResources()
    .getBoolean(R.bool.bAdvancedFeaturesEnabled);
```
Working with Integer Resources

- In addition to strings and Boolean values, you can also store integers as resources.
- Integer resources are defined in XML under the res/values/ project directory.
- They are compiled into the application package at build time.
Defining Integer Resources in XML

- Integer values are appropriately tagged with the `<integer>` tag and represent a name/value pair.
  - The `name` attribute is how you refer to the specific integer programmatically, so name these resources wisely.

- Here’s an example of the integer resource file:
  `res/values/nums.xml`:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <integer name="numTimesToRepeat">25</integer>
  <integer name="startingAgeOfCharacter">3</integer>
</resources>
```
Using Integer Resources Programmatically

- To use the integer resource, you must load it using the Resources class.
- The following code accesses your application’s integer resource named numTimesToRepeat:

  ```java
  int repTimes = getResources()
                  .getInteger(R.integer.numTimesToRepeat);
  ```
- You can create integer arrays as resources using the `<integer-array>` tag.
  - Child `<item>` tags define one for each item in the array.
- You can then load the integer array using the `getIntArray()` method of the Resources class.
Working with Colors

- Android applications can store RGB color values.
  - These can be applied to screen elements.
- You can use these values to set the color of text or other elements, such as the screen background.
- Color resources are defined in XML under the res/values/ project directory and compiled into the application package at build time.
Defining Color Resources in XML

- RGB color values always start with the hash symbol (#).
- The alpha value can be given for transparency control.
- The following color formats are supported:
  - #RGB (#F00 is 12-bit color, red)
  - #ARGB (#8F00 is 12-bit color, red with alpha 50%)
  - #RRGGBB (#FF00FF is 24-bit color, magenta)
  - #AARRGGBB (#80FF00FF is 24-bit color, magenta, with alpha 50%)
- Color values are appropriately tagged with the <color> tag and represent a name/value pair.
- Here's an example of a simple color resource file res/values/colors.xml:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="background_color">#006400</color>
  <color name="text_color">#FFE4C4</color>
</resources>
```
Using Color Resources Programmatically

- The example at the beginning of the chapter accessed a color resource.
- Color resources are simply integers.
- The following example shows the method getColor() retrieving a color resource called prettyTextColor:

```java
int myResourceColor = getResources()
    .getColor(R.color.prettyTextColor);
```
Working with Dimensions

- Many user interface layout controls, such as text controls and buttons, are drawn to specific dimensions.
- These dimensions can be stored as resources.
- Dimension values always end with a unit of measurement tag.
Defining Dimension Resources in XML

- Dimension values are tagged with the `<dimen>` tag and represent a name/value pair.
- They are defined in XML under the `res/values/` project directory and compiled into the application package at build time.
## Defining Dimension Resources in XML (Cont’d)

<table>
<thead>
<tr>
<th>Unit of Measurement</th>
<th>Description</th>
<th>Resource Tag Required</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels</td>
<td>Actual screen pixels</td>
<td>px</td>
<td>20px</td>
</tr>
<tr>
<td>Inches</td>
<td>Physical measurement</td>
<td>in</td>
<td>1in</td>
</tr>
<tr>
<td>Millimeters</td>
<td>Physical measurement</td>
<td>mm</td>
<td>1mm</td>
</tr>
<tr>
<td>Points</td>
<td>Common font measurement unit</td>
<td>pt</td>
<td>14pt</td>
</tr>
<tr>
<td>Screen density–independent pixels</td>
<td>Pixels relative to 160dpi screen (preferable for dimension screen compatibility)</td>
<td>dp</td>
<td>1dp</td>
</tr>
<tr>
<td>Scale-independent pixels</td>
<td>Best for scalable font display</td>
<td>sp</td>
<td>14sp</td>
</tr>
</tbody>
</table>
Defining Dimension Resources in XML (Cont’d)

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="FourteenPt">14pt</dimen>
    <dimen name="OneInch">1in</dimen>
    <dimen name="TenMillimeters">10mm</dimen>
    <dimen name="TenPixels">10px</dimen>
</resources>
```
Using Dimension Resources Programmatically

- Dimension resources are simply floating-point values.
- The `getDimension()` method retrieves a dimension resource called `textPointSize`:

```java
float myDimension = getResources()
    .getDimension(R.dimen.textPointSize);
```
## Drawable Resources

<table>
<thead>
<tr>
<th>Drawable Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShapeDrawable</td>
<td>A geometric shape such as a circle or rectangle</td>
</tr>
<tr>
<td>ScaleDrawable</td>
<td>Defines the scaling of a drawable</td>
</tr>
<tr>
<td>TransitionDrawable</td>
<td>Used to cross-fade between drawables</td>
</tr>
<tr>
<td>ClipDrawable</td>
<td>Drawable used to clip a region of a drawable</td>
</tr>
<tr>
<td>StateListDrawable</td>
<td>Used to define different states of a drawable such as pressed or selected</td>
</tr>
<tr>
<td>LayerDrawable</td>
<td>An array of drawables</td>
</tr>
<tr>
<td>BitmapDrawable</td>
<td>Bitmap graphics file</td>
</tr>
<tr>
<td>NinePatchDrawable</td>
<td>Stretchable PNG file</td>
</tr>
</tbody>
</table>
Working with Simple Drawables

- You can specify simple colored rectangles by using the drawable resource type.
  - This can be applied to other screen elements.
- These drawable types are defined in specific paint colors, much as the color resources are defined.
Defining Simple Drawable Resources in XML

- Simple paintable drawable resources are defined in XML under the `res/values/` project directory and compiled into the application package at build time.
- Paintable drawable resources use the `<drawable>` tag and represent a name/value pair.
- Here’s an example of a simple drawable resource file called `res/values/drawables.xml`:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <drawable name="red_rect">#F00</drawable>
</resources>
```
You can also create XML files that describe other Drawable subclasses, such as ShapeDrawable.

Drawable XML definition files are stored in the res/drawable/ directory within your project, along with image files. This is not the same as storing <drawable> resources, which are paintable drawables.

ShapeDrawable resources are stored in the res/values directory, as explained previously.

Here's a simple ShapeDrawable described in the file res/drawable/red_oval.xml:

```xml
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
       android:shape="oval">
  <solid android:color="#f00"/>
</shape>
```
Using Simple Drawable Resources Programmatically

- Drawable resources defined with `<drawable>` are simply rectangles of a given color, which is represented by the Drawable subclass `ColorDrawable`.
- The following code retrieves a `ColorDrawable` resource called `redDrawable`:

```java
ColorDrawable myDraw =
    (ColorDrawable) getResources().getDrawable(R.drawable.redDrawable);
```
## Working with Images

<table>
<thead>
<tr>
<th>Supported Image Format</th>
<th>Description</th>
<th>Required Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable Network Graphics (PNG)</td>
<td>Preferred format (lossless)</td>
<td>.png</td>
</tr>
<tr>
<td>Nine-Patch Stretchable Graphics</td>
<td>Preferred format (lossless)</td>
<td>.9.png</td>
</tr>
<tr>
<td>Joint Photographic Experts Group (JPEG)</td>
<td>Acceptable format (lossy)</td>
<td>.jpeg,.jpg</td>
</tr>
<tr>
<td>Graphics Interchange Format (GIF)</td>
<td>Discouraged format</td>
<td>.gif</td>
</tr>
<tr>
<td>WebP (WEBP)</td>
<td>Android 4.0+</td>
<td>.webp</td>
</tr>
</tbody>
</table>
These image formats are all well supported by popular graphics editors such as Adobe Photoshop, GIMP, and Microsoft Paint.

- The Nine-Patch Stretchable Graphics can be created from PNG files using the `draw9patch` tool included with the Android SDK.
  - This tool is found under the `tools` directory.

- Adding image resources to your project is easy.
  - Simply drag the image asset into the `res/drawable/` resource directory hierarchy.
Working with Nine-Patch Stretchable Graphics

- It can be handy to use stretchable graphics to allow a single graphic that can scale appropriately for different screen sizes and orientations or different lengths of text.
- This can save you or your designer a lot of time in creating graphics for many different screen sizes.
- Android supports Nine-Patch Stretchable Graphics for this purpose.
  - Nine-Patch graphics are PNG graphics that have patches, or areas of the image, defined to scale appropriately, instead of the entire image being scaled as one unit.
  - Often, the center segment is transparent or a solid color for a background because it’s the stretched part.
  - A common use for Nine-Patch graphics is to create frames and borders.
  - Little more than the corners are needed, so a very small graphics file can be used to frame any size image or View control.
- Nine-Patch Stretchable Graphics can be created from PNG files using the draw9patch tool included with the tools directory of the SDK.
Using Image Resources Programmatically

- Image resources are another kind of Drawable called a BitmapDrawable.
- Most of the time, you need only the resource ID of the image to set as an attribute on a user interface control.
- For example:
  - If we drop the graphics file flag.png into the res/drawable directory and add an ImageView control to the main layout, we can interact with that control programmatically in the layout by first using the findViewById() method to retrieve a control by its identifier and then casting it to the proper type of control—in this case, an ImageView (android.widget.ImageView) object:

```java
ImageView flagImageView = (ImageView) findViewById(R.id.ImageView01);
flagImageView.setImageResource(R.drawable.flag);
```
If you want to access the `BitmapDrawable` (`android.graphics.drawable.BitmapDrawable`) object directly, you can request that resource directly using the `getDrawable()` method, as follows:

```java
BitmapDrawable bitmapFlag = (BitmapDrawable) getResources().getDrawable(R.drawable.flag);
int iBitmapHeightInPixels = bitmapFlag.getIntrinsicHeight();
int iBitmapWidthInPixels = bitmapFlag.getIntrinsicWidth();
```
Using Image Resources Programmatically (Cont’d)

- If you work with Nine-Patch graphics, the call to `getDrawable()` will return a `NinePatchDrawable` object instead of a `BitmapDrawable` object:

```java
NinePatchDrawable stretchy = (NinePatchDrawable) getResources().getDrawable(R.drawable.pyramid);
int iStretchyHeightInPixels =
    stretchy.getIntrinsicHeight();
int iStretchyWidthInPixels =
    stretchy.getIntrinsicWidth();
```
Working with Color State Lists

- A special resource type called `<selector>` can be used to define different colors or drawables to be used depending on a control’s state.
- For example, you could define a color state list for a Button control: gray when the button is disabled, green when it is enabled, and yellow when it is being pressed.
- Similarly, you could provide different drawables based on the state of an ImageButton control.
- The `<selector>` element can have one or more child `<item>` elements that define different colors for different states.
- There are quite a few attributes that you are able to define for the `<item>` element, and you can define one or more for supporting many different states for your View objects.
## Working with Color State Lists (Cont’d)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Required attribute for specifying a hexadecimal color in one of these formats: #RGB, #ARGB, #RRGGBB, #AARRGGBB, where A is alpha, R is red, G is green, and B is blue.</td>
</tr>
<tr>
<td>state_enabled</td>
<td>Boolean denoting whether this object is capable of receiving touch or click events, true or false.</td>
</tr>
<tr>
<td>state_checked</td>
<td>Boolean denoting checked or unchecked, true or false.</td>
</tr>
<tr>
<td>state_checkable</td>
<td>Boolean denoting checkable or not checkable, true or false.</td>
</tr>
<tr>
<td>state_selected</td>
<td>Boolean denoting selected or not selected, true or false.</td>
</tr>
<tr>
<td>state_focused</td>
<td>Boolean denoting focused or not focused, true or false.</td>
</tr>
<tr>
<td>state_pressed</td>
<td>Boolean denoting pressed or not pressed, true or false.</td>
</tr>
</tbody>
</table>
Defining a Color State List Resource

- You first must create a resource file defining the various states that you want to apply to your View object.
- To do so, you define a color resource that contains the `<selector>` element and the various `<item>`s and attributes that you want to apply.
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:state_disabled="true"
        android:color="#C0C0C0"/>
  <item android:state_enabled="true"
        android:color="#00FF00"/>
  <item android:state_pressed="true"
        android:color="#FFFF00"/>
  <item android:color="#000000"/>
</selector>
Defining a Button for Applying the State List Resource

- Now that we have a color state list resource, we can apply this value to one of our View objects.
- Below, we define a Button and set the textColor attribute to the state list resource file text_color.xml that we defined previously.

```xml
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/text"
    android:textColor="@color/text_color" />
```
Working with Animation

- Android supports several kinds of animation.
- Two of the simplest varieties are:
  - Frame-by-frame
    - Frame-by-frame animation involves the display of a sequence of images in rapid succession.
  - Tweening
    - Tweened animation involves applying standard graphical transformations such as rotations and fades on a single image.
- The Android SDK provides some helper utilities for loading and using animation resources.
- These are found in android.view.animation.AnimationUtils class.
Defining and Using Frame-by-Frame Animation Resources

- Frame-by-frame animation is often used when the content changes from frame to frame.
- This can be used for complex frame transitions—much like a kid’s flip-book.
- To define frame-by-frame resources, take the following steps:
  1. Save each frame graphic as an individual drawable resource.
     - It may help to name your graphics sequentially, in the order in which they are displayed—for example, frame1.png, frame2.png, and so on.
  2. Define the animation set resource in an XML file within the res/drawable/ resource directory hierarchy.
  3. Load, start, and stop the animation programmatically.
<?xml version="1.0" encoding="utf-8" ?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/splash1"
        android:duration="500" />
    <item android:drawable="@drawable/splash2"
        android:duration="500" />
    <item android:drawable="@drawable/splash3"
        android:duration="500" />
</animation-list>
Defining and Using Frame-by-Frame Animation Resources (Cont’d)

- Frame-by-frame animation set resources defined with `<animation-list>` are represented by the Drawable subclass `AnimationDrawable`.

- The following code retrieves an `AnimationDrawable` resource called `juggle`:

  ```java
  AnimationDrawable jugglerAnimation = 
  (AnimationDrawable) getResources().
 getDrawable(R.drawable.juggle);
  ```

- After you have a valid `AnimationDrawable` (android.graphics.drawable.AnimationDrawable), you can assign it to a `View` control on the screen and start and stop animation.
Defining and Using Tweened Animation Resources

- Tweened animation features include:
  - Scaling
  - Fading
  - Rotation
  - Translation

- These actions can be applied simultaneously or sequentially and might use different interpolators.

- Tweened animation sequences are not tied to a specific graphics file, so you can write one sequence and then use it for a variety of different graphics.

- For example:
  - You can make moon, star, and diamond graphics all pulse using a single scaling sequence, or you can make them spin using a rotate sequence.
Defining Tweened Animation Sequence Resources in XML

- Graphic animation sequences can be stored as specially formatted XML files in the `res/anim/` directory and are compiled into the application binary at build time.

- Here’s an example of a simple animation resource file called `res/anim/spin.xml` that defines a simple rotate operation—rotating the target graphic counterclockwise four times in place, taking 10 seconds to complete:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:shareInterpolator="false">
    <rotate
        android:fromDegrees="0"
        android:toDegrees="-1440"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="10000" />  
</set>
```
Using Tweened Sequence Resources Programmatically

- If we go back to the earlier example of a BitmapDrawable, we can now include some animation simply by adding the following code to load the animation resource file spin.xml and set the animation in motion:

```java
ImageView flagImageView = (ImageView)findViewById(R.id.ImageView01);
flagImageView.setImageResource(R.drawable.flag);
...
Animation an = AnimationUtils.loadAnimation(this, R.anim.spin);
flagImageView.startAnimation(an);
```

- Now you have your graphic spinning.
- Notice that we loaded the animation using the base class object Animation.
You can also extract specific animation types using the subclasses that match:

- RotateAnimation
- ScaleAnimation
- TranslateAnimation
- AlphaAnimation

These are found in the android.view.animation package.

There are a number of different interpolators you can use with your tweened animation sequences.
Working with Menus

- You can also include menu resources in your project files.
- Like animation resources, menu resources are not tied to a specific control but can be reused in any menu control.
- Menu resources (which are sets of individual menu items) are stored as specially formatted XML files in the `res/menu/` directory and are compiled into the application package at build time.
Defining Menu Resources in XML

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/start"
    android:title="Start!"
    android:orderInCategory="1"></item>
  <item android:id="@+id/stop" android:title="Stop!"
    android:orderInCategory="4"></item>
  <item android:id="@+id/accel"
    android:title="Vroom! Accelerate!"
    android:orderInCategory="2"></item>
  <item android:id="@+id/decel"
    android:title="Decelerate!"
    android:orderInCategory="3"></item>
</menu>
```
Using Menu Resources Programmatically

- To access the preceding menu resource called res/menu/speed.xml, simply override the method onCreateOptionsMenu() in your Activity class, returning true to cause the menu to be displayed:
  ```java
  public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.speed, menu);
      return true;
  }
  ```
- If you run your application and press the Menu button, you see the menu.
- A number of other XML attributes can be assigned to menu items.
Working with XML Files

- You can include arbitrary XML resource files to your project.
- You should store these XML files in the `res/xml/` directory, and they are compiled into the application package at build time.
- The Android SDK has a variety of packages and classes available for XML manipulation.
Defining Raw XML Resources

- Put a simple XML file in the res/xml/ directory. In this case, the file my_pets.xml with the following contents can be created:

```xml
<?xml version="1.0" encoding="utf-8"?>
<pets>
    <pet name="Bit" type="Bunny" />
    <pet name="Nibble" type="Bunny" />
    <pet name="Stack" type="Bunny" />
    <pet name="Queue" type="Bunny" />
    <pet name="Heap" type="Bunny" />
    <pet name="Null" type="Bunny" />
    <pet name="Nigiri" type="Fish" />
    <pet name="Sashimi II" type="Fish" />
    <pet name="Kiwi" type="Lovebird" />
</pets>
```
Using XML Resources Programmatically

- Now you can access this XML file as a resource programmatically in the following manner:
  ```java
  XmlResourceParser myPets =
    getResources().getXml(R.xml.my_pets);
  ```
- You can then use the parser of your choice to parse the XML.
Working with Raw Files

- Your application can also include raw files as part of its resources.
- For example:
  - Your application might use raw files such as audio files, video files, and other file formats not supported by the Android Resource Packaging Tool (aapt).
Defining Raw File Resources

- All raw resource files are included in the `res/raw/` directory and are added to your package without further processing.
You can access raw file resources from the `res/raw/` resource directory and any resource from the `res/drawable/` directory (bitmap graphics files, anything not using the `<resource>` XML definition method).

Here’s one way to open a file called `the_help.txt`:

```java
InputStream iFile = getResources()
    .openRawResource(R.raw.the_help);
```
References to Resources

- You can reference resources instead of duplicating them.
- For example:
  - Your application might want to reference a single string resource in multiple string arrays.
- The most common use of resource references is in layout XML files, where layouts can reference any number of resources to specify attributes for layout colors, dimensions, strings, and graphics.
  - Another common use is within style and theme resources.
- Resources are referenced using the following format:
  - @resource_type/variable_name
References to Resources (Cont’d)

- Recall that earlier we had a string array of soup names.
- If we want to localize the soup listing, a better way to create the array is to create individual string resources for each soup name and then store the references to those string resources in the string array (instead of the text).
- To do this, we define the string resources in the `res/strings.xml` file like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Application Name</string>
    <string name="chicken_soup">Organic Chicken Noodle</string>
    <string name="minestrone_soup">Veggie Minestrone</string>
    <string name="chowder_soup">New England Clam Chowder</string>
</resources>
```
Then, we can define a localizable string array that references the string resources by name in the res/arrays.xml file like this:

```xml
<resources>
    <string-array name="soups">
        <item>@string/minestrone_soup</item>
        <item>@string/chowder_soup</item>
        <item>@string/chicken_soup</item>
    </string-array>
</resources>
```
You can also use references to make aliases to other resources. For example:

- You can alias the system resource for the `OK` string to an application resource name by including the following in your `strings.xml` resource file:

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string d="app_ok">@android:string/ok</string>
</resources>
```
Working with Layouts

- Much as Web designers use HTML, user interface designers can use XML to define Android application screen elements and layout.
- A layout XML resource is where many different resources come together to form the definition of an Android application screen.
- Layout resource files are included in the `res/layout/` directory and are compiled into the application package at build time.
- Layout files might include many user interface controls and define the layout for an entire screen or describe custom controls used in other layouts.
Working with Layouts (Cont’d)

Android says hello world!
Working with Layouts (Cont’d)

- The activity_simple_resource_view.xml layout file that displays this screen references a number of other resources, including colors, strings, and dimension values, all of which were defined in the strings.xml, styles.xml, colors.xml, and dimens.xml resource files.
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/background_color">
    <TextView android:id="@+id/TextView01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/test_string"
        android:textColor="@color/text_color"
        android:gravity="center"
        android:textSize="@dimen/text_size" />
</LinearLayout>
Designing Layouts in Android Studio

- Layouts can be designed and previewed in Android Studio using the resource editor functionality.
- If you click the project file `res/layout/activity_simple_resource_view.xml`, you see the Design tab, which shows both a preview of how the `activity_simple_resource_view.xml` will appear on a device, and the Text tab, which shows the raw XML of the layout file.
Designing Layouts in Android Studio (Cont’d)
As with most user interface editors, Android Studio works well for basic layout needs, enables you to create user interface controls such as TextView and Button controls easily, and lets you set the controls’ properties in the Properties pane.
Now is a great time to get to know the layout editor.

- Create a new Android project called ParisView (available as a sample project).
- Navigate to the res/layout/activity_paris_view.xml layout file and double-click it to open it.
  - By default, it’s simple with only a black rectangle and a string of text.
  - To the right of the Design preview of Android Studio, you notice the Component Tree section.
  - This outline is the XML hierarchy of this layout file.
- By default, you see a LinearLayout.
  - If you expand it, you see it contains one TextView control.
  - Click the TextView control.
    - You see that the Properties section of Android Studio now has all the properties available for that object.
    - If you scroll down to the property called text, you see that it’s set to the string resource variable @string/hello_world.
You can use the layout editor to set and preview layout control properties.

For example:
- You can modify the TextView property called `textSize` by typing `18pt` (a dimension).
- You see the results of your change to the property immediately in the preview area.

Take a moment to switch to the `TextView`.
- You notice that the properties you set are now in the XML.
Designing Layouts in Android Studio (Cont’d)
Now, within the Palette found under the Widgets section, drag and drop the ImageView object within the preview editor.

Now you have a new control in your layout.

Drag two PNG (or JPG) graphics files into your res/drawable/project directory, naming them flag.png and background.png.

Now, go to the Component Tree to make sure the ImageView is selected, then browse to the properties of your ImageView control, and set the src property manually by typing @drawable/flag.

Now you see that the graphic shows up in your preview.

While we’re at it, select the LinearLayout object and set its background property to the background drawable you added.
Using Layout Resources Programmatically

- Objects within layouts, whether Button or ImageView controls, are all derived from the View class.
- To retrieve a TextView object named TextView01, called in an Activity class after the call to setContentView():

```java
TextView txt =
    (TextView) findViewById(R.id.TextView01);
```
- You can also access the underlying XML of a layout resource much as you would any XML file.
- To retrieve the main.xml layout file for XML parsing:

```java
XmlResourceParser myMainXml = getResources().
    getLayout(R.layout.activity_paris_view);
```
- Developers can also define custom layouts with unique attributes.
Referencing System Resources

- In addition to the resources included in your project, you can also take advantage of the generic resources provided as part of the Android SDK.
- You can access system resources much as you would your own resources.
- The `android` package contains all kinds of resources, which you can browse by looking in the `android.R` subclasses.
- Here, you find system resources for the following:
  - Animation sequences for fading in and out
  - Arrays of email/phone types (home, work, and such)
  - Standard system colors
  - Dimensions for application thumbnails and icons
  - Many commonly used drawable and layout types
  - Error strings and standard button text
  - System styles and themes
You can reference system resources in other resources such as layout files by specifying the @android package name before the resource.

For example:
- To set the background to the system color for darker gray, you set the appropriate background color attribute to @android:color/darker_gray.

You can access system resources programmatically through the android.R class.
- If we go back to our animation example, we could have used a system animation instead of defining our own.

Here is the same animation example again, except it uses a system animation to fade in:

```java
ImageView flagImageView = (ImageView)findViewById(R.id.ImageView01);
flagImageView.setImageResource(R.drawable.flag);
Animation an = AnimationUtils.loadAnimation(this, android.R.anim.fade_in);
flagImageView.startAnimation(an);
```
Chapter 6
Summary

- We have learned what Android resources are.
- We have learned how to set simple resource values using Android Studio.
- We are now able to work with many different types of Android resources.
- We are now able to create layouts and screen elements for our applications.
- We have learned how to take advantage of referencing generic system resources.
References and More Information

- Android API Guides: “App Resources”:

- Android API Guides: “Resource Types”:
  - http://d.android.com/guide/topics/resources/available-resources.html