Creating Resources

Application resources are stored under the res/ folder of your project hierarchy. In this folder, each of the available resource types can have a subfolder containing its resources.

If you start a project using the ADT wizard, it will create a res folder that contains subfolders for the values, drawable, and layout resources that contain the default layout, application icon, and string resource defi nitions, respectively, as shown in Figure 3-4.

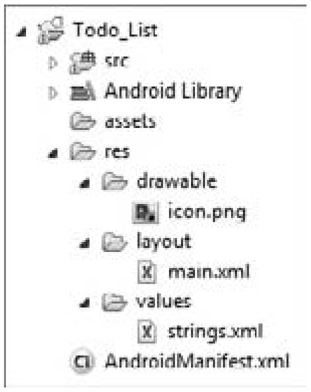

Figure 3-4

There are seven primary resource types that have different folders: simple values, drawables, layouts, animations, XML, styles, and raw resources. When your application is built, these resources will be compiled as effi ciently as possible and included in your application package.

This process also creates an R class fi le that contains references to each of the resources you include in your project. This lets you reference the resources in your code, with the advantage of design time syntax checking.

The following sections describe the specifi c resource types available within these categories and how to create them for your applications.

In all cases, the resource fi lenames should contain only lowercase letters, numbers, and the period (.) and underscore (_) symbols.

Creating Simple Values

Supported simple values include strings, colors, dimensions, and string or integer arrays. All simple values are stored within XML fi les in the res/values folder.

Within each XML fi le, you indicate the type of value being stored using tags as shown in the sample XML fi le below:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">To Do List</string>
<color name="app_background">#FF0000FF</color>
<dimen name="default_border">5px</dimen>
<array name="string_array">
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
</array>
<array name="integer_array">
<item>3</item>
<item>2</item>
<item>1</item>
</array>
</resources>
```

This example includes all of the simple value types. By convention, resources are separated into separate fi les for each type; for example, res/values/strings.xml would contain only string resources.

The following sections detail the options for defi ning simple resources.

Strings

Externalizing your strings helps maintain consistency within your application and makes it much easier to create localized versions.
String resources are specifi ed using the string tag as shown in the following XML snippet:

```
<string name=”stop_message”>Stop.</string>
```

Android supports simple text styling, so you can use the HTML tags <b>, <i>, and <u> to apply bold, italics, or underlining to parts of your text strings as shown in the example below:

```
<string name=”stop_message”><b>Stop.</b></string>
```

You can use resource strings as input parameters for the String.format method. However, String.format does not support the text styling described above. To apply styling to a format string, you have to escape the HTML tags when creating your resource, as shown below:

```
<string name=”stop_message”>&lt;b>Stop&lt;/b>. %1$s</string>
```

Within your code, use the Html.fromHtml method to convert this back into a styled character sequence:

```
String rString = getString(R.string.stop_message);
String fString = String.format(rString, “Collaborate and listen.”);
CharSequence styledString = Html.fromHtml(fString);
```

Colors

Use the color tag to defi ne a new color resource. Specify the color value using a # symbol followed by the (optional) alpha channel, then the red, green, and blue values using one or two hexadecimal numbers with any of the following notations:

- ❑ #RGB
- ❑ #RRGGBB
- ❑ #ARGB
- ❑ #ARRGGBB

The following example shows how to specify a fully opaque blue and a partially transparent green:

```
<color name=”opaque_blue”>#00F</color>
<color name=”transparent_green”>#7700FF00</color>
```

Dimensions

Dimensions are most commonly referenced within style and layout resources. They’re useful for creating layout constants such as borders and font heights.

To specify a dimension resource, use the dimen tag, specifying the dimension value, followed by an identifi er describing the scale of your dimension:

- ❑ px Screen pixels
- ❑ in Physical inches
- ❑ pt Physical points
- ❑ mm Physical millimeters
- ❑ dp Density-independent pixels relative to a 160-dpi screen
- ❑ sp Scale-independent pixels

These alternatives let you defi ne a dimension not only in absolute terms, but also using relative scales that account for different screen resolutions and densities to simplify scaling on different hardware.
The following XML snippet shows how to specify dimension values for a large font size and a standard border:

```
<dimen name=”standard_border”>5px</dimen>
<dimen name=”large_font_size”>16sp</dimen>
```

Styles and Themes

Style resources let your applications maintain a consistent look and feel by specifying the attribute values used by Views. The most common use of themes and styles is to store the colors and fonts for an application.
You can easily change the appearance of your application by simply specifying a different style as the theme in your project manifest.
To create a style, use a style tag that includes a name attribute, and contains one or more item tags. Each item tag should include a name attribute used to specify the attribute (such as font size or color) being defi ned. The tag itself should then contain the value, as shown in the skeleton code below:

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
<style name=”StyleName”>
<item name=”attributeName”>value</item>
```

```
</style>
</resources>
```

Styles support inheritance using the parent attribute on the style tag, making it easy to create simple variations.
The following example shows two styles that can also be used as a theme; a base style that sets several text properties and a second style that modifi es the fi rst to specify a smaller font.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="BaseText">
<item name="android:textSize">14sp</item>
<item name="android:textColor">#111</item>
</style>
<style name="SmallText" parent="BaseText">
<item name="android:textSize">8sp</item>
</style>
</resources>
```

Drawables

Drawable resources include bitmaps and NinePatch (stretchable PNG) images. They are stored as individual files in the res/drawable folder.

The resource identifi er for a bitmap resource is the lowercase fi lename without an extension.
The preferred format for a bitmap resource is PNG, although JPG and GIF fi les are also supported.
NinePatch (or stretchable) images are PNG fi les that mark the parts of an image that can be stretched. NinePatch images must be properly defi ned PNG fi les that end in .9.png. The resource identifi er for NinePatches is the fi lename without the trailing .9.png.

A *NinePatch* is a variation of a PNG image that uses a 1-pixel border to defi ne the area of the image that can be stretched if the image is enlarged. To create a NinePatch, draw single-pixel black lines that represent stretchable areas along the left and top borders of your image. The unmarked sections won't be resized, and the relative size of each of the marked sections will remain the same as the image size changes.

NinePatches are a powerful technique for creating images for the backgrounds of Views or Activities that may have a variable size; for example, Android uses NinePatches for creating button backgrounds.

Layouts

Layout resources let you decouple your presentation layer by designing user-interface layouts in XML rather than constructing them in code.

The most common use of a layout is to defi ne the user interface for an Activity. Once defi ned in XML, the layout is "infl ated" within an Activity using setContentView, usually within the onCreate method.

You can also reference layouts from within other layout resources, such as layouts for each row in a List View. More detailed information on using and creating layouts in Activities can be found in Chapter 4.

Using layouts to create your screens is best-practice UI design in Android. The decoupling of the layout from the code lets you create optimized layouts for different hardware confi gurations, such as varying screen sizes, orientation, or the presence of keyboards and touch screens.

Each layout defi nition is stored in a separate fi le, each containing a single layout, in the res/layout folder. The fi lename then becomes the resource identifi er.

A thorough explanation of layout containers and View elements is included in the next chapter, but as an example, the following code snippet shows the layout created by the New Project Wizard. It uses a LinearLayout as a layout container for a TextView that displays the "Hello World" greeting.

```
<?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">
```

```
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Hello World!"
/>
</LinearLayout>
```

Animations

Android supports two types of animation. *Tweened animations* can be used to rotate, move, stretch, and fade a View; or you can create *frame-by-frame animations* to display a sequence of drawable images.

A comprehensive overview of creating, using, and applying animations can be found in Chapter 11. Defi ning animations as external resources allows you to reuse the same sequence in multiple places and provides you with the opportunity to present an alternative animation based on device hardware or orientation.

Tweened Animations

Each tweened animation is stored in a separate XML fi le in the project's res/anim folder. As with layouts and drawable resources, the animation's fi lename is used as its resource identifi er.
An animation can be defi ned for changes in alpha (fading), scale (scaling), translate (moving), or rotate (rotating).

Each of these animation types supports attributes to defi ne what the sequence will do:

Alpha	fromAlpha and toAlpha	Float from 0 to 1
Scale	fromXScale/toXScale	Float from 0 to 1
	fromYScale/toYScale	Float from 0 to 1
	pivotX/pivotY	String of the percentage of graphic width/height from 0% to 100%
Translate	fromX/toX	Float from 0 to 1
	fromY/toY	Float from 0 to 1
Rotate	fromDegrees/toDegrees	Float from 0 to 360
	pivotX/pivotY	String of the percentage of graphic width/height from 0% to 100%

You can create a combination of animations using the set tag. An animation set contains one or more animation transformations and supports various additional tags and attributes to customize when and how each animation within the set is run.

The following list shows some of the set tags available:

❑ **duration** Duration of the animation in milliseconds.
❑ **startOffset** Millisecond delay before starting this animation.
❑ **fi llBefore** True to apply the animation transformation before it begins.
❑ **fi llAfter** True to apply the animation transformation after it begins.
❑ **interpolator** Interpolator to set how the speed of this effect varies over time. Chapter 11

explores the interpolators available. To specify an interpolator, reference the system animation resources at android:anim/interpolatorName.
If you do not use the startOffset *tag, all the animation effects within a set will execute simultaneously.*
The following example shows an animation set that spins the target 360 degrees while it shrinks and fades out:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/accelerate_interpolator">
<rotate
android:fromDegrees="0"
android:toDegrees="360"
android:pivotX="50%"
android:pivotY="50%"
android:startOffset="500"
android:duration="1000" />
<scale
android:fromXScale="1.0"
android:toXScale="0.0"
android:fromYScale="1.0"
```

```
android:toYScale=”0.0”
android:pivotX=”50%”
android:pivotY=”50%”
android:startOffset=”500”
android:duration=”500” />
<alpha
android:fromAlpha=”1.0”
android:toAlpha=”0.0”

android:startOffset=”500”
android:duration=”500” />
</set>
```

Frame-by-Frame Animations

Frame-by-frame animations let you create a sequence of drawables, each of which will be displayed for a specifi ed duration, on the background of a View.

Because frame-by-frame animations represent animated drawables, they are stored in the res/drawble folder, rather than with the tweened animations, and use their fi lenames (without the xml extension) as
their resource IDs.

The following XML snippet shows a simple animation that cycles through a series of bitmap resources, displaying each one for half a second. In order to use this snippet, you will need to create new image resources rocket1 through rocket3.

```
<animation-list
xmlns:android=”http://schemas.android.com/apk/res/android”
android:oneshot=”false”>
<item android:drawable=”@drawable/rocket1” android:duration=”500” />
<item android:drawable=”@drawable/rocket2” android:duration=”500” />
<item android:drawable=”@drawable/rocket3” android:duration=”500” />
</animation-list>
```