# Using Resources

As well as the resources you create, Android supplies several system resources that you can use in your applications. The resources can be used directly from your application code and can also be referenced from within other resources (e.g., a dimension resource might be referenced in a layout defi nition).

Later in this chapter, you'll learn how to defi ne alternative resource values for different languages, locations, and hardware. It's important to note that when using resources you cannot choose a particular specialized version. Android will automatically select the most appropriate value for a given resource identifi er based on the current hardware and device settings.

## Using Resources in Code

You access resources in code using the static R class. R is a generated class based on your external resources and created by compiling your project. The R class contains static subclasses for each of the resource types for which you've defi ned at least one resource. For example, the default new project includes the R.string and R.drawable subclasses.

*If you are using the ADT plug-in in Eclipse, the* R *class will be created automatically when you make any change to an external resource fi le or folder. If you are not using the plug-in, use the AAPT tool to compile your project and generate the* R *class.* R *is a compiler-generated class, so don't make any manual modifi cations to it as they will be lost when the fi le is regenerated.*

Each of the subclasses within R exposes its associated resources as variables, with the variable names matching the resource identifi ers — for example, R.string.app_name or R.drawable.icon.

The value of these variables is a reference to the corresponding resource's location in the resource table, *not* an instance of the resource itself.

Where a constructor or method, such as setContentView, accepts a resource identifi er, you can pass in the resource variable, as shown in the code snippet below:

```
// Inflate a layout resource.
setContentView(R.layout.main);

// Display a transient dialog box that displays the
// error message string resource.
Toast.makeText(this, R.string.app_error, Toast.LENGTH_LONG).show();
```

When you need an instance of the resource itself, you'll need to use helper methods to extract them from the resource table, represented by an instance of the Resources class.

Because these methods perform lookups on the application's resource table, these helper methods can't be static. Use the getResources method on your application context as shown in the snippet below to access your application's Resource instance:

```
Resources myResources = getResources();
```

The Resources class includes getters for each of the available resource types and generally works by passing in the resource ID you'd like an instance of. The following code snippet shows an example of using the helper methods to return a selection of resource values:

```
Resources myResources = getResources();
CharSequence styledText = myResources.getText(R.string.stop_message);
Drawable icon = myResources.getDrawable(R.drawable.app_icon);
int opaqueBlue = myResources.getColor(R.color.opaque_blue);
float borderWidth = myResources.getDimension(R.dimen.standard_border);
Animation tranOut;
tranOut = AnimationUtils.loadAnimation(this, R.anim.spin_shrink_fade);
String[] stringArray;
stringArray = myResources.getStringArray(R.array.string_array);
int[] intArray = myResources.getIntArray(R.array.integer_array);
```

Frame-by-frame animated resources are infl ated into AnimationResources. You can return the value using getDrawable and casting the return value as shown below:

```
AnimationDrawable rocket;
rocket = (AnimationDrawable)myResources.getDrawable(R.drawable.frame_by_frame);
```

**At the time of going to print, there is a bug in the** AnimationDrawable **class. Currently, AnimationDrawable resources are not properly loaded until some time after an Activity's** onCreate **method has completed. Current work-arounds use timers to force a delay before loading a frame-by-frame resource.**

## *Referencing Resources in Resources*

You can also reference resources to use as attribute values in other XML resources.

This is particularly useful for layouts and styles, letting you create specialized variations on themes and localized strings and graphics. It's also a useful way to support different images and spacing for a layout to ensure that it's optimized for different screen sizes and resolutions.

To reference one resource from another, use @ notation, as shown in the following snippet:
attribute="@[packagename:]resourcetype/resourceidentifier"

*Android will assume you're using a resource from the same package, so you only need to fully qualify the package name if you're using a resource from a different package.*

The following snippet creates a layout that uses color, dimension, and string resources:

```
<?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:padding="@dimen/standard_border">
<EditText
android:id="@+id/myEditText"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/stop_message"
android:textColor="@color/opaque_blue"
/>
</LinearLayout>
```

## *Using System Resources*

The native Android applications externalize many of their resources, providing you with various strings, images, animations, styles, and layouts to use in your applications.

Accessing the system resources in code is similar to using your own resources. The difference is that you use the native android resource classes available from android.R, rather than the application-specifi c R class. The following code snippet uses the getString method available in the application context to retrieve an error message available from the system resources:

```
CharSequence httpError = getString(android.R.string.httpErrorBadUrl);
```

To access system resources in XML, specify *Android* as the package name, as shown in this XML snippet:

```
<EditText
android:id="@+id/myEditText"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@android:string/httpErrorBadUrl"
android:textColor="@android:color/darker_gray"
/>
```

## *Referring to Styles in the Current Theme*

Themes are an excellent way to ensure consistency for your application's UI. Rather than fully defi ne each style, Android provides a shortcut to let you use styles from the currently applied theme. To do this, you use ?android: rather than @ as a prefi x to the resource you want to use. The following example shows a snippet of the above code but uses the current theme's text color rather than an external resource:

```
<EditText
android:id="@+id/myEditText"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/stop_message"
android:textColor="?android:textColor" />
```

This technique lets you create styles that will change if the current theme changes, without having to modify each individual style resource.

## *To-Do List Resources Example*

In this example, you'll create new external resources in preparation for adding functionality to the To-Do List example you started in Chapter 2. The string and image resources you create here will be used in Chapter 4 when you implement a menu system for the To-Do List application.

The following steps will show you how to create text and icon resources to use for the add and remove menu items, and how to create a theme to apply to the application:

1. Create two new PNG images to represent adding, and removing, a to-do list item. Each image should have dimensions of approximately 16 · 16 pixels, like those illustrated in Figure 3-5.

Figure 3-5

2. Copy the images into your project's res/drawable folder, and refresh your project. Your project hierarchy should appear as shown in Figure 3-6.

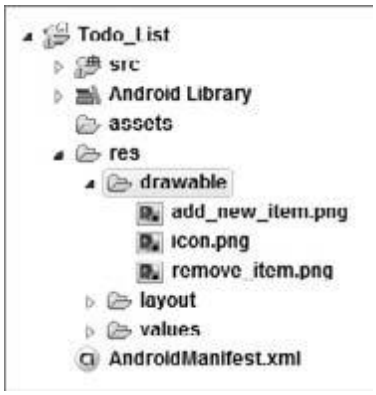


Figure 3-6

3. Open the strings.xml resource from the res/values folder, and add values for the "add_new," "remove," and "cancel" menu items. (You can remove the default "hello" string value while you're there.)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">To Do List</string>
<string name="add_new">Add New Item</string>
<string name="remove">Remove Item</string>
<string name="cancel">Cancel</string>
</resources>
```

4. Create a new theme for the application by creating a new styles.xml resource in the res/values folder. Base your theme on the standard Android theme, but set values for a default text size.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="ToDoTheme" parent="@android:style/Theme.Black">
<item name="android:textSize">12sp</item>
</style>
</resources>
```

5. Apply the theme to your project in the manifest:

```
<activity android:name=".ToDoList"
android:label="@string/app_name"
android:theme="@style/ToDoTheme">
```

# *Creating Resources for Different Languages and Hardware*

One of the most powerful reasons to externalize your resources is Android's dynamic resource selection mechanism.

Using the structure described below, you can create different resource values for specifi c languages, locations, and hardware confi gurations that Android will choose between dynamically at run time. This lets you create language-, location-, and hardware-specifi c user interfaces without having to change your code.

Specifying alternative resource values is done using a parallel directory structure within the res/ folder, using hyphen (-) separated text qualifi ers to specify the conditions you're supporting. The example hierarchy below shows a folder structure that features default string values, along with a French language alternative with an additional Canadian location variation:

Project/
res/
values/
strings.xml
values-fr/
strings.xml
values-fr-rCA/
strings.xml

The following list gives the available qualifi ers you can use to customize your resource fi les:

- **Language** Using the lowercase two-letter ISO 639-1 language code (e.g., en)
- **Region** A lowercase "r" followed by the uppercase two-letter ISO 3166-1-alpha-2 language code (e.g., rUS, rGB)
- **Screen Orientation** One of port (portrait), land (landscape), or square (square)
- **Screen Pixel Density** Pixel density in dots per inch (dpi) (e.g., 92dpi, 108dpi)
- **Touchscreen Type** One of notouch, stylus, or finger
- **Keyboard Availability** Either of keysexposed or keyshidden
- **Keyboard Input Type** One of nokeys, qwerty, or 12key
- **UI Navigation Type** One of notouch, dpad, trackball, or wheel
- **Screen Resolution** Screen resolution in pixels with the largest dimension fi rst (e.g., 320x240)

You can specify multiple qualifi ers for any resource type, separating each qualifi er with a hyphen. Any combination is supported; however, they must be used in the order given in the list above, and no more than one value can be used per qualifier.

The following example shows valid and invalid directory names for alternative drawable resources.

- **Valid:**

drawable-en-rUS
drawable-en-keyshidden
drawable-land-notouch-nokeys-320x240

- **Invalid:**

drawable-rUS-en (out of order)
drawable-rUS-rUK (multiple values for a single qualifier)

When Android retrieves a resource at run time, it will fi nd the best match from the available alternatives. Starting with a list of all the folders in which the required value exists, it then selects the one with the greatest number of matching qualifi ers. If two folders are an equal match, the tiebreaker will be based on the order of the matched qualifi ers in the above list.