3.2 APP RESOURCES

You should always externalize resources, such as images and strings, from your application code, so that you can maintain them indepen- dently. You should place each type of resource in a specific subdirectory of your project's *res/* directory, as shown in Fig. 3.2.

drawable/ file contains bitmap files, such as *png*, *jpg* and *gif*. layout/ contains XML files that define a user interface layout. menu/ contains XML files that define application menus.

vaules/ contains XML files that contain simple values, such as strings, integers, and colors.

Besides the ones shown in Fig. 3.1, we can add some other resource files into *res/* directory, such as *animator/*, *raw/*, and *xml/* files.

The animator file contains Android property animations. The property animation system is a robust framework that allows you to animate almost anything. You can define an animation to change any object property over time, regardless of whether it draws to the screen or not. You can find more information about property animation at http://developer.android.com/guide/topics/graphics/prop-animation.html.

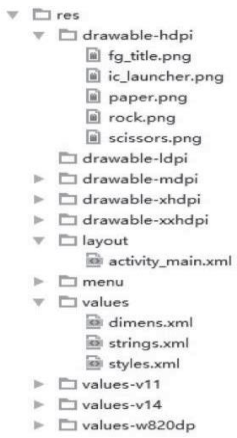

Figure 3.2 File hierarchy for a simple project.

The raw file stores any files in their raw form. You must call *Re-*

source.openRawResource() to open these resources with a raw *Input- Stream.*

The XML file contains arbitrary XML files that can be read at run- time by calling *Resource.getXML()*. Various XML configuration files must be saved here, such as a searchable configuration.

HINT: Never save resource files directly inside the res/ directory, because it will cause a compiler error.

3.3 APP MAINFEST

The manifest file is indispensable in every Android application. The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code. Among other things, the manifest does the following:

1. It names the Java package for the application. The package name serves as a unique identifier for the application.

2. It describes the components of the application, the activities, services, broadcast receivers, and content providers that the appli-cation is composed of. It names the classes that implement each of the components and publishes their capabilities (for example, which Intent messages they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched.

3. It determines which processes will host application components.

4. It declares which permissions the application must have in or- der to access protected parts of the Application Programming Interface (API) and interact with other applications.

5. It also declares the permissions that others are required to have in order to interact with the application's components.
6. It lists the Instrumentation classes that provide profiling and other information as the application is running. These declarations are present in the manifest only while the application is being developed and tested; they're removed before the application is published.
7. It declares the minimum level of the Android API that the application requires.
8. It lists the libraries that the application must be linked against.

3.3.1 Elements

Only the <manifest> and <application> elements are required, they each must be present and can occur only once. Most of the others can, occur many times or not at all, although at least some of them must be present for the manifest to accomplish anything meaningful.

If an element contains anything at all, it contains other elements. All values are set through attributes, not as character data within an element. Elements at the same level are generally not ordered. For example,

<activity>, <provider>, and <service> elements can be intermixed in any sequence. (An <activity-alias> element is the exception to this rule: It must follow the <activity> it is an alias for.)

3.3.2 Attributes

In a formal sense, all attributes are optional. However, there are some that must be specified for an element to accomplish its purpose. Use the documentation as a guide. For truly optional attributes, it mentions a default value or states what happens in the absence of a specification. Except for some attributes of the root <manifest> element, all attribute names begin with an android: prefix, for example, an- droid:alwaysRetainTaskState. Because the prefix is universal, the documentation generally omits it when referring to attributes by name.

3.3.3 Declaring Class Names

Many elements correspond to Java objects, including elements for the application itself (the <application> element) and its principal compo- nents, activities (<activity>), services (<service>), broadcast receivers (<receiver>), and content providers (<provider>).

If you define a subclass, as you almost always would for the component classes (Activity, Service, BroadcastReceiver, and Content- Provider), the subclass is declared through a name attribute. The name must include the full package designation. However, as a shorthand, if the first character of the string is a period, the string is appended to the application's package name (as specified by the <manifest> element's package attribute). When starting a component, Android creates an instance of the named subclass. If a subclass isn't specified, it creates an instance of the base class.