Introducing the Application Manifest

Each Android project includes a manifest fi le, AndroidManifest.xml, stored in the root of the project hierarchy. The manifest lets you defi ne the structure and metadata of your application and its components.

It includes nodes for each of the components (Activities, Services, Content Providers, and Broadcast Receivers) that make up your application and, using Intent Filters and Permissions, determines how they interact with each other and other applications.

It also offers attributes to specify application metadata (like its icon or theme), and additional top-level nodes can be used for security settings and unit tests as described below.

The manifest is made up of a root manifest tag with a package attribute set to the project's package. It usually includes an xmlns:android attribute that supplies several system attributes used within the fi le. A typical manifest node is shown in the XML snippet below:

```
<manifest xmlns:android=http://schemas.android.com/apk/res/android
package="com.my_domain.my_app">
[ ... manifest nodes ... ]
</manifest>
```

The manifest tag includes nodes that defi ne the application components, security settings, and test classes that make up your application. The following list gives a summary of the available manifest node tags, and an XML snippet demonstrating how each one is used:

❑ application A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme). It also acts as a container that includes the Activity, Service, Content Provider, and Broadcast Receiver tags used to specify the application components.

```
<application android:icon="@drawable/icon"
android:theme="@style/my_theme">
[ ... application nodes ... ]
</application>
```

❑ activity An activity tag is required for every Activity displayed by your application, using the android:name attribute to specify the class name. This must include the main launch Activity and any other screen or dialogs that can be displayed. Trying to start an Activity that's not defi ned in the manifest will throw a runtime exception. Each Activity node supports intent-filter child tags that specify which Intents launch the Activity.

```
<activity android:name=".MyActivity" android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

❑ service As with the activity tag, create a new service tag for each Service class used in your application. (Services are covered in detail in Chapter 8.) Service tags also support intent-filter child tags to allow late runtime binding.

```
<service android:enabled="true" android:name=".MyService"></service>
```

❑ provider Provider tags are used for each of your application's Content Providers. Content Providers are used to manage database access and sharing within and between applications and are examined in Chapter 6.

```
<provider android:permission="com.paad.MY_PERMISSION"
android:name=".MyContentProvider"
android:enabled="true"
android:authorities="com.paad.myapp.MyContentProvider">
</provider>
```

❑ receiver By adding a receiver tag, you can register a Broadcast Receiver without having to launch your application fi rst. As you'll see in Chapter 5, Broadcast Receivers are like global event listeners that, once registered, will execute whenever a matching

Intent is broadcast by an application. By registering a Broadcast Receiver in the manifest, you can make this process entirely autonomous. If a matching Intent is broadcast, your application will be started automatically and the registered Broadcast Receiver will be run.

```
<receiver android:enabled="true"
android:label="My Broadcast Receiver"
android:name=".MyBroadcastReceiver">
</receiver>
```

❑ uses-permission As part of the security model, uses-permission tags declare the permissions you've determined that your application needs for it to operate properly. The permissions you include will be presented to the user, to grant or deny, during installation. Permissions are required for many of the native Android services, particularly those with a cost or security implication (such as dialing, receiving SMS, or using the location-based services). As shown in the item below, third-party applications, including your own, can also specify permissions before providing access to shared application components.

```
<uses-permission android:name="android.permission.ACCESS_LOCATION">
</uses-permission>
```

❑ permission Before you can restrict access to an application component, you need to defi ne a permission in the manifest. Use the permission tag to create these permission defi nitions. Application components can then require them by adding the android:permission attribute. Other applications will then need to include a uses-permission tag in their manifests (and have it granted) before they can use these protected components. Within the permission tag, you can specify the level of access the permission will permit (normal, dangerous, signature, signatureOrSystem), a label, and an external resource containing the description that explain the risks of granting this permission.

```
<permission android:name="com.paad.DETONATE_DEVICE"
android:protectionLevel="dangerous"
android:label="Self Destruct"
android:description="@string/detonate_description">
</permission>
```

❑ instrumentation Instrumentation classes provide a framework for running tests on your Activities and Services at run time. They provide hooks to monitor your application and its interaction with the system resources. Create a new node for each of the test classes you've created
for your application.

```
<instrumentation android:label="My Test"
android:name=".MyTestClass"
android:targetPackage="com.paad.aPackage">
</instrumentation>
```

A more detailed description of the manifest and each of these nodes can be found at http://code.google.com/android/devel/bblocks-manifest.html The ADT New Project Wizard automatically creates a new manifest fi le when it creates a new project. You'll return to the manifest as each of the application components is introduced.