The AndroidManifest.xml File

Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. Among other things, the manifest does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application — the activities, services, broadcast receivers, and content providers that the application 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.
- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It also declares the permissions that others are required to have in order to interact with the application's components.
- 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.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

Structure of the Manifest File

The diagram below shows the general structure of the manifest file and every element that it can contain. Each element, along with all of its attributes, is documented in full in a separate file. To view detailed information about any element, click on the element name in the diagram, in the alphabetical list of elements that follows the diagram, or on any other mention of the element name.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
```

```
    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>

        <uses-library />

    </application>

</manifest>
```

All the elements that can appear in the manifest file are listed below in alphabetical order. These are the only legal elements; you cannot add your own elements or attributes.

File Conventions

Some conventions and rules apply generally to all elements and attributes in the manifest:

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.)

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, `android:alwaysRetainTaskState`. Because the prefix is universal, the documentation generally omits it when referring to attributes by name.

Declaring class names

Many elements correspond to Java objects, including elements for the application itself (the <application> element) and its principal components — 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 ContentProvider), the subclass is declared through a `name` attribute. The name must include the full package designation. For example, an Service subclass might be declared as follows:

```
<manifest . . . >
    <application . . . >
        <service android:name="com.example.project.SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

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). The following assignment is the same as the one above:

```
<manifest package="com.example.project" . . . >
    <application . . . >
        <service android:name=".SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

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.

Multiple values

If more than one value can be specified, the element is almost always repeated, rather than listing multiple values

within a single element. For example, an intent filter can list several actions:

```
<intent-filter . . . >
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    . . .
</intent-filter>
```

Resource values

Some attributes have values that can be displayed to users — for example, a label and an icon for an activity. The values of these attributes should be localized and therefore set from a resource or theme. Resource values are expressed in the following format,

```
@[package:]type:name
```

where the *package* name can be omitted if the resource is in the same package as the application, *type* is a type of resource — such as "string" or "drawable" — and *name* is the name that identifies the specific resource. For example:

```
<activity android:icon="@drawable/smallPic" . . . >
```

Values from a theme are expressed in a similar manner, but with an initial '?' rather than '@':

```
?[package:]type:name
```

String values

Where an attribute value is a string, double backslashes ('\\') must be used to escape characters — for example, '\\n' for a newline or '\\uxxxx' for a Unicode character.

File Features

The following sections describe how some Android features are reflected in the manifest file.

Intent Filters

The core components of an application (its activities, services, and broadcast receivers) are activated by *intents*. An intent is a bundle of information (an `Intent` object) describing a desired action — including the data to be acted upon, the category of component that should perform the action, and other pertinent instructions. Android locates an appropriate component to respond to the intent, launches a new instance of the component if one is needed, and passes it the Intent object.

Components advertise their capabilities — the kinds of intents they can respond to — through *intent filters*. Since the Android system must learn which intents a component can handle before it launches the component, intent filters are specified in the manifest as `<intent-filter>` elements. A component may have any number of filters, each one describing a different capability.

An intent that explicitly names a target component will activate that component; the filter doesn't play a role. But an intent that doesn't specify a target by name can activate a component only if it can pass through one of the component's filters.

For information on how Intent objects are tested against intent filters, see a separate document, Intents and Intent Filters.

Icons and Labels

A number of elements have `icon` and `label` attributes for a small icon and a text label that can be displayed to users. Some also have a `description` attribute for longer explanatory text that can also be shown on-screen. For example, the `<permission>` element has all three of these attributes, so that when the user is asked whether to grant the

permission to an application that has requested it, an icon representing the permission, the name of the permission, and a description of what it entails can all be presented to the user.

In every case, the icon and label set in a containing element become the default `icon` and `label` settings for all of the container's subelements. Thus, the icon and label set in the `<application>` element are the default icon and label for each of the application's components. Similarly, the icon and label set for a component — for example, an `<activity>` element — are the default settings for each of the component's `<intent-filter>` elements. If an `<application>` element sets a label, but an activity and its intent filter do not, the application label is treated as the label for both the activity and the intent filter.

The icon and label set for an intent filter are used to represent a component whenever the component is presented to the user as fulfilling the function advertised by the filter. For example, a filter with "`android.intent.action.MAIN`" and "`android.intent.category.LAUNCHER`" settings advertises an activity as one that initiates an application — that is, as one that should be displayed in the application launcher. The icon and label set in the filter are therefore the ones displayed in the launcher.

Permissions

A *permission* is a restriction limiting access to a part of the code or to data on the device. The limitation is imposed to protect critical data and code that could be misused to distort or damage the user experience.

Each permission is identified by a unique label. Often the label indicates the action that's restricted. For example, here are some permissions defined by Android:

```
android.permission.CALL_EMERGENCY_NUMBERS
android.permission.READ_OWNER_DATA
android.permission.SET_WALLPAPER
android.permission.DEVICE_POWER
```

A feature can be protected by at most one permission.

If an application needs access to a feature protected by a permission, it must declare that it requires that permission with a `<uses-permission>` element in the manifest. Then, when the application is installed on the device, the installer determines whether or not to grant the requested permission by checking the authorities that signed the application's certificates and, in some cases, asking the user. If the permission is granted, the application is able to use the protected features. If not, its attempts to access those features will simply fail without any notification to the user.

An application can also protect its own components (activities, services, broadcast receivers, and content providers) with permissions. It can employ any of the permissions defined by Android (listed in `android.Manifest.permission`) or declared by other applications. Or it can define its own. A new permission is declared with the `<permission>` element. For example, an activity could be protected as follows:

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . . />
    <uses-permission android:name="com.example.project.DEBIT_ACCT" />
    . . .
    <application . . .>
        <activity android:name="com.example.project.FreneticActivity"
                  android:permission="com.example.project.DEBIT_ACCT"
                  . . . >
            . . .
        </activity>
    </application>
</manifest>
```

Note that, in this example, the `DEBIT_ACCT` permission is not only declared with the `<permission>` element, its use is also requested with the `<uses-permission>` element. Its use must be requested in order for other components of the application to launch the protected activity, even though the protection is imposed by the application itself.

If, in the same example, the `permission` attribute was set to a permission declared elsewhere (such as `android.permission.CALL_EMERGENCY_NUMBERS`, it would not have been necessary to declare it again with a `<permission>` element. However, it would still have been necessary to request its use with `<uses-permission>`.

The <permission-tree> element declares a namespace for a group of permissions that will be defined in code. And <permission-group> defines a label for a set of permissions (both those declared in the manifest with <permission> elements and those declared elsewhere). It affects only how the permissions are grouped when presented to the user. The <permission-group> element does not specify which permissions belong to the group; it just gives the group a name. A permission is placed in the group by assigning the group name to the <permission> element's permissionGroup attribute.

Libraries

Every application is linked against the default Android library, which includes the basic packages for building applications (with common classes such as Activity, Service, Intent, View, Button, Application, ContentProvider, and so on).

However, some packages reside in their own libraries. If your application uses code from any of these packages, it must explicitly asked to be linked against them. The manifest must contain a separate <uses-library> element to name each of the libraries. (The library name can be found in the documentation for the package.)

↑ Go to top