App Manifest Overview

Every app project must have an `AndroidManifest.xml` file (with precisely that name) at the root of the project source set (https://developer.android.com/studio/build/index.html#sourcesets). The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

Among many other things, the manifest file is required to declare the following:

- The app's package name, which usually matches your code's namespace. The Android build tools use this to determine the location of code entities when building your project. When packaging the app, the build tools replace this value with the application ID from the Gradle build files, which is used as the unique app identifier on the system and on Google Play. Read more about the package name and app ID (#package-name).
- The components of the app, which include all activities, services, broadcast receivers, and content providers. Each component must define basic properties such as the name of its Kotlin or Java class. It can also declare capabilities such as which device configurations it can handle, and intent filters that describe how the component can be started. Read more about app components (#components).
- The permissions that the app needs in order to access protected parts of the system or other apps. It also declares any permissions that other apps must have if they want to access content from this app. Read more about permissions (#perms).
- The hardware and software features the app requires, which affects which devices can install the app from Google Play. Read more about device compatibility (#compatibility).

If you're using Android Studio (https://developer.android.com/studio/) to build your app, the manifest file is created for you, and most of the essential manifest elements are added as you build your app (especially when using code templates (https://developer.android.com/studio/projects/templates.html)).

File features

The following sections describe how some of the most important characteristics of your app are reflected in the manifest file.

Package name and application ID

The manifest file's root element requires an attribute for your app's package name (usually matching your project directory structure—the Java namespace).

For example, the following snippet shows the root `<manifest>` (https://developer.android.com/guide/topics/manifest/manifest-element.html) element with the package name "`com.example.myapp`":

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
</manifest>
```

While building your app into the final application package (APK), the Android build tools use the `package` attribute for two things:

- It applies this name as the namespace for your app's generated `R.java` class (used to access your app resources (https://developer.android.com/guide/topics/resources/overview.html)).

  Example: With the above manifest, the `R` class is created at `com.example.myapp.R`.
- It uses this name to resolve any relative class names that are declared in the manifest file.

  Example: With the above manifest, an activity declared as `<activity android:name=".MainActivity">` is resolved to be `com.example.myapp.MainActivity`.

As such, the name in the manifest's `package` attribute should always match your project's base package name where you keep your activities and other app code. Of course, you can have other sub-packages in your project, but those files must import the `R.java` class using the namespace from the `package` attribute.

However, beware that, once the APK is compiled, the `package` attribute also represents your app's universally unique application ID. After the build tools perform the above tasks based on the `package` name, they replace the `package` value with the value given to the `applicationId` property in your project's `build.gradle` file (used in Android Studio projects). This final value for the `package` attribute must be universally unique because it is the only guaranteed way to identify your app on the system and in Google Play.

The distinction between the **`package`** name in the manifest and the **`applicationId`** in the **`build.gradle`** file can be a bit confusing. But if you keep them same, you have nothing to worry about.

However, if you decide to make your code's namespace (and thus, the **`package`** name in the manifest) something different from the **`applicationId`** from the build file, be sure you fully understand the implications of setting the application ID (https://developer.android.com/studio/build/application-id.html). That page explains how you can safely adjust the manifest's **`package`** name independent of the build file's **`applicationId`**, and change the application ID for different build configurations.

App components

For each app component (https://developer.android.com/guide/components/fundamentals.html#Components) that you create in your app, you must declare a corresponding XML element in the manifest file:

- `<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html) for each subclass of `Activity` (https://developer.android.com/reference/android/app/Activity.html).
- `<service>` (https://developer.android.com/guide/topics/manifest/service-element.html) for each subclass of `Service` (https://developer.android.com/reference/android/app/Service.html).
- `<receiver>` (https://developer.android.com/guide/topics/manifest/receiver-element.html) for each subclass of `BroadcastReceiver` (https://developer.android.com/reference/android/content/BroadcastReceiver.html).
- `<provider>` (https://developer.android.com/guide/topics/manifest/provider-element.html) for each subclass of `ContentProvider` (https://developer.android.com/reference/android/content/ContentProvider.html).

If you subclass any of these components without declaring it in the manifest file, the system cannot start it.

The name of your subclass must be specified with the **`name`** attribute, using the full package designation. For example, an `Activity` (https://developer.android.com/reference/android/app/Activity.html) subclass can be declared as follows:

```
<manifest ... >
    <application ... >
        <activity android:name="com.example.myapp.MainActivity" ... >
        </activity>
    </application>
</manifest>
```

However, if the first character in the `name` value is a period, the app's package name (from the `<manifest>` (https://developer.android.com/guide/topics/manifest/manifest-element.html) element's `package` (https://developer.android.com/guide/topics/manifest/manifest-element.html#package) attribute) is prefixed to the name. For example, the following activity name is resolved to `"com.example.myapp.MainActivity"`:

```
<manifest package="com.example.myapp" ... >
    <application ... >
        <activity android:name=".MainActivity" ... >
            ...
        </activity>
    </application>
</manifest>
```

If you have app components that reside in sub-packages (such as in `com.example.myapp.purchases`), the `name` value must add the missing sub-package names (such as `".purchases.PayActivity"`) or use the fully-qualified package name.

Intent filters

App activities, services, and broadcast receivers are activated by *intents*. An intent is a message defined by an `Intent` (https://developer.android.com/reference/android/content/Intent.html) object that describes an action to perform, including the data to be acted upon, the category of component that should perform the action, and other instructions.

When an app issues an intent to the system, the system locates an app component that can handle the intent based on *intent filter* declarations in each app's manifest file. The system launches an instance of the matching component and passes the `Intent` (https://developer.android.com/reference/android/content/Intent.html) object to that component. If more than one app can handle the intent, then the user can select which app to use.

An app component can have any number of intent filters (defined with the `<intent-filter>` (https://developer.android.com/guide/topics/manifest/intent-filter-element.html) element), each one describing a different capability of that component.

For more information, see the Intents and Intent Filters (https://developer.android.com/guide/components/intents-filters.html) document.

Icons and labels

A number of manifest elements have `icon` and `label` attributes for displaying a small icon and a text label, respectively, to users for the corresponding app component.

In every case, the icon and label that are set in a parent element become the default `icon` and `label` value for all child elements. For example, the icon and label that are set in the `<application>` (https://developer.android.com/guide/topics/manifest/application-element.html) element are the default icon and label for each of the app's components (such as all activities).

The icon and label that are set in a component's `<intent-filter>` (https://developer.android.com/guide/topics/manifest/intent-filter-element.html) are shown to the user whenever that component is presented as an option to fulfill an intent. By default, this icon is inherited from whichever icon is declared for the parent component (either the `<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html) or `<application>` (https://developer.android.com/guide/topics/manifest/application-element.html) element), but you might want to change the icon for an intent filter if it provides a unique action that you'd like to better indicate in the chooser dialog. For more information, see Allow Other Apps to Start Your Activity (https://developer.android.com/training/basics/intents/filters.html).

Permissions

Android apps must request permission to access sensitive user data (such as contacts and SMS) or certain system features (such as the camera and internet access). Each permission is identified by a unique label. For example, an app that needs to send SMS messages must have the following line in the manifest:

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

Beginning with Android 6.0 (API level 23), the user can approve or reject some app permisions at runtime. But no matter which Android version your app supports, you must declare all permission requests with a `<uses-permission>` (https://developer.android.com/guide/topics/manifest/uses-permission-element.html) element in the manifest. If the permission is granted, the app is able to use the protected features. If not, its attempts to access those features fail.

Your app can also protect its own components with permissions. It can use any of the permissions that are defined by Android, as listed in `android.Manifest.permission` (https://developer.android.com/reference/android/Manifest.permission.html), or a permission that's declared in another app. Your app can also define its own permissions. A new permission is declared with the `<permission>` (https://developer.android.com/guide/topics/manifest/permission-element.html) element.

For more information, see the Permissions Overview (https://developer.android.com/guide/topics/permissions/overview.html).

Device compatibility

The manifest file is also where you can declare what types of hardware or software features your app requires, and thus, which types of devices your app is compatible with. Google Play Store does not allow your app to be installed on devices that don't provide the features or system version that your app requires.

There are several manifest tags that define which devices your app is compatible with. The following are just a couple of the most common tags.

<uses-feature>

The `<uses-feature>` (https://developer.android.com/guide/topics/manifest/uses-feature-element.html) element allows you to declare hardware and software features your app needs. For example, if your app cannot achieve basic functionality on a device without a compass sensor, you can declare the compass sensor as required with the following manifest tag:

```
<manifest ... >
    <uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true" />
    ...
</manifest>
```

Note: If you'd like to make your app available on Chromebooks, there are some important hardware and software feature limitations that you should consider. For more information, see App Manifest Compatibility for Chromebooks (https://developer.android.com/topic/arc/manifest.html).

<uses-sdk>

Each successive platform version often adds new APIs not available in the previous version. To indicate the minimum version with which your app is compatible, your manifest must include the `<uses-sdk>` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html) tag and its `minSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#min) attribute.

However, beware that attributes in the `<uses-sdk>` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html) element are overridden by corresponding properties in the `build.gradle` (https://developer.android.com/studio/build/index.html#build-files) file. So if you're using Android Studio, you must specify the `minSdkVersion` and `targetSdkVersion` values there instead:

```
android {
  defaultConfig {
    applicationId 'com.example.myapp'

    // Defines the minimum API level required to run the app.
    minSdkVersion 15

    // Specifies the API level used to test the app.
    targetSdkVersion 26

    ...
  }
}
```

For more information about the `build.gradle` file, read about how to configure your build (https://developer.android.com/studio/build/index.html).

To learn more about how to declare your app's support for different devices, see the Device Compatibility Overview (https://developer.android.com/guide/practices/compatibility.html).

File conventions

This section describes the conventions and rules that generally apply to all elements and attributes in the manifest file.

Elements

Only the `<manifest>` (https://developer.android.com/guide/topics/manifest/manifest-element.html) and `<application>`

(https://developer.android.com/guide/topics/manifest/application-element.html) elements are required. They each must occur only once. Most of the other elements can occur zero or more times. However, some of them must be present to make the manifest file useful.

All of the values are set through attributes, not as character data within an element.

Elements at the same level are generally not ordered. For example, the `<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html), `<provider>` (https://developer.android.com/guide/topics/manifest/provider-element.html), and `<service>` (https://developer.android.com/guide/topics/manifest/service-element.html) elements can be placed in any order. There are two key exceptions to this rule:

- An `<activity-alias>` (https://developer.android.com/guide/topics/manifest/activity-alias-element.html) element must follow the `<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html) for which it is an alias.
- The `<application>` (https://developer.android.com/guide/topics/manifest/application-element.html) element must be the last element inside the `<manifest>` (https://developer.android.com/guide/topics/manifest/manifest-element.html) element.

Attributes

Technically, all attributes are optional. However, many attributes must be specified so that an element can accomplish its purpose. For truly optional attributes, the reference documentation (#reference) indicates the default values.

Except for some attributes of the root `<manifest>` (https://developer.android.com/guide/topics/manifest/manifest-element.html) 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.

Multiple values

If more than one value can be specified, the element is almost always repeated, rather than multiple values being listed 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 are displayed to users, such as the title for an activity or your app icon. The value for these attributes might differ based on the user's language or other device configurations (such as to provide a different icon size based on the device's pixel density), so the values should be set from a resource or theme, instead of hard-coded into the manifest file. The actual value can then change based on alternative resources (https://developer.android.com/guide/topics/resources/providing-resources.html) that you provide for different device configurations.

Resources are expressed as values with the following format:

"`@[`***package***`:]`***type***`/`***name***"

You can omit the ***package*** name if the resource is provided by your app (including if it is provided by a library dependency, because library resources are merged into yours (https://developer.android.com/studio/write/add-resources.html#resource_merging)). The only other valid package name is `android`, when you want to use a resource from the Android framework.

The ***type*** is a type of resource, such as `string` (https://developer.android.com/guide/topics/resources/string-resource.html) or `drawable` (https://developer.android.com/guide/topics/resources/drawable-resource.html), and the ***name*** is the name that identifies the specific resource. Here is an example:

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

For more information about how to add resources in your project, read Providing Resources (https://developer.android.com/guide/topics/resources/providing-resources.html).

To instead apply a value that's defined in a theme (https://developer.android.com/guide/topics/ui/look-and-feel/themes.html), the first character must be `?` instead of `@`:

"`?[`***package***`:]`***type***`/`***name***"

String values

Where an attribute value is a string, you must use double backslashes (`\\`) to escape characters, such as `\\n` for a newline or `\\uxxxx` for a Unicode character.

Manifest elements reference

The following table provides links to reference documents for all valid elements in the `AndroidManifest.xml` file.

`<action>` (https://developer.android.com/guide/topics/manifest/action-element.html)

`<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html)

`<activity-alias>` (https://developer.android.com/guide/topics/manifest/activity-alias-element.html)

`<application>` (https://developer.android.com/guide/topics/manifest/application-element.html)

`<category>` (https://developer.android.com/guide/topics/manifest/category-element.html)

`<compatible-screens>` (https://developer.android.com/guide/topics/manifest/compatible-screens-el

`<data>` (https://developer.android.com/guide/topics/manifest/data-element.html)

`<grant-uri-permission>` (https://developer.android.com/guide/topics/manifest/grant-uri-permissio

`<instrumentation>` (https://developer.android.com/guide/topics/manifest/instrumentation-element.h

`<intent-filter>` (https://developer.android.com/guide/topics/manifest/intent-filter-element.html)

`<manifest>` (https://developer.android.com/guide/topics/manifest/manifest-element.html)

`<meta-data>` (https://developer.android.com/guide/topics/manifest/meta-data-element.html)

`<path-permission>` (https://developer.android.com/guide/topics/manifest/path-permission-element.l

`<permission>` (https://developer.android.com/guide/topics/manifest/permission-element.html)

`<permission-group>` (https://developer.android.com/guide/topics/manifest/permission-group-eleme

`<permission-tree>` (https://developer.android.com/guide/topics/manifest/permission-tree-element.h

`<provider>` (https://developer.android.com/guide/topics/manifest/provider-element.html)

`<receiver>` (https://developer.android.com/guide/topics/manifest/receiver-element.html)

`<service>` (https://developer.android.com/guide/topics/manifest/service-element.html)

`<supports-gl-texture>` (https://developer.android.com/guide/topics/manifest/supports-gl-texture-e

---

`<supports-screens>` (https://developer.android.com/guide/topics/manifest/supports-screens-elemei

---

`<uses-configuration>` (https://developer.android.com/guide/topics/manifest/uses-configuration-ele

---

`<uses-feature>` (https://developer.android.com/guide/topics/manifest/uses-feature-element.html)

`<uses-library>` (https://developer.android.com/guide/topics/manifest/uses-library-element.html)

`<uses-permission>` (https://developer.android.com/guide/topics/manifest/uses-permission-element.

`<uses-permission-sdk-23>` (https://developer.android.com/guide/topics/manifest/uses-permission

---

`<uses-sdk>` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html)

Example manifest file

The XML below is a simple example `AndroidManifest.xml` that declares two activities for the app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.example.myapp">

    <!-- Beware that these values are overridden by the build.gradle file -->
    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="26" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- This name is resolved to com.example.myapp.MainActivity
             based upon the package attribute -->
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".DisplayMessageActivity"
            android:parentActivityName=".MainActivity" />
    </application>
</manifest>
```

Twitter
Follow @AndroidDev on Twitter

Google+
Follow Android Developers on Google+

YouTube
Check out Android Developers on YouTube