<manifest>

syntax:

```
<manifest xmlns:android (#nspace)="http://schemas.android.com/apk/res
          package (#package)="string"
          android:sharedUserId (#uid)="string"
          android:sharedUserLabel (#uidlabel)="string resource"
          android:versionCode (#vcode)="integer"
          android:versionName (#vname)="string"
          android:installLocation (#install)=["auto" | "internalOnly" | "pref
    . . .
</manifest>
```

contained in:

none

must contain:

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

can contain:

<compatible-screens>
(https://developer.android.com/guide/topics/manifest/compatible-screens-element.html)
<instrumentation>
(https://developer.android.com/guide/topics/manifest/instrumentation-element.html)
<permission>
(https://developer.android.com/guide/topics/manifest/permission-element.html)
<permission-group>
(https://developer.android.com/guide/topics/manifest/permission-group-element.html)
<permission-tree>
(https://developer.android.com/guide/topics/manifest/permission-tree-element.html)
<supports-gl-texture>
(https://developer.android.com/guide/topics/manifest/supports-gl-texture-element.html)
<supports-screens>
(https://developer.android.com/guide/topics/manifest/supports-screens-element.html)
<uses-configuration>
(https://developer.android.com/guide/topics/manifest/uses-configuration-element.html)

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

description:

The root element of the AndroidManifest.xml file. It must contain an `<application>` (https://developer.android.com/guide/topics/manifest/application-element.html) element and specify `xmlns:android` and `package` attributes.

attributes:

`xmlns:android`

Defines the Android namespace. This attribute should always be set to "`http://schemas.android.com/apk/res/android`".

`package`

A full Java-language-style package name for the Android app. The name may contain uppercase or lowercase letters ('A' through 'Z'), numbers, and underscores ('_'). However, individual package name parts may only start with letters.
While building your app into the an application package (APK), the build system uses 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)).

  For example, if `package` is set to "`com.example.myapp`", 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.
  For example, if `package` is set to "`com.example.myapp`", an activity declared as `<activity android:name=".MainActivity">` is resolved to be `com.example.myapp.MainActivity`.

This name is also the default name for your app process (see the `<application>` element's `process`

(https://developer.android.com/guide/topics/manifest/application-element.html#aff) attribute). And it's the default task affinity for your activities (see the `<activity>` element's `taskAffinity` (https://developer.android.com/guide/topics/manifest/activity-element.html#aff) attribute).

This name also represents the application ID, which must be universally unique in order to publish your app in Google Play (https://developer.android.com/distribute/google-play/index.html). However, toward the end of the APK build process, the build tools override the **`package`** name using the **`applicationId`** property from the **`build.gradle`** file (used by Android Studio projects). As long as you keep the manifest's **`package`** name the same as the build file's **`applicationId`**, this won't be a concern. But if these two values differ, you should understand the differences between the "package name" and "application ID" by reading how to set the application ID (https://developer.android.com/studio/build/application-id.html).

To avoid conflicts with other developers, you should use Internet domain ownership as the basis for your package names (in reverse). For example, apps published by Google start with **`com.google`**.

Note: Both the `com.example` and `com.android` namespaces are forbidden by Google Play.

If you want to change your package name after you publish your app, you can, but **you must keep the `applicationId` the same**. The `applicationId` defines the unique identity for your app on Google Play. So if you change it, the APK is considered to be a different app and users of the previous version will not receive an update. For more information, see how to set the application ID (https://developer.android.com/studio/build/application-id.html).

`android:sharedUserId`

The name of a Linux user ID that will be shared with other apps. By default, Android assigns each app its own unique user ID. However, if this attribute is set to the same value for two or more apps, they will all share the same ID — provided that their certificate sets are identical. Apps with the same user ID can access each other's data and, if desired, run in the same process.

`android:targetSandboxVersion`

The target sandbox for this app to use. The higher the sandbox version number, the higher the level of security. Its default value is `1`; you can also set it to `2`.

Setting this attribute to 2 switches the app to a different SELinux sandbox.

The following restrictions apply to a level 2 sandbox:

- The default value of **`usesCleartextTraffic`** (https://developer.android.com/guide/topics/manifest/application-element.html#usesCleartextTraffic) in the Network Security Config is false.
- Uid sharing is not permitted.

For Android Instant Apps targeting Android 8.0 (API level 26) or higher, this attribute must be set to 2. You can set the sandbox level in the installed version of your app to the less restrictive level 1, but if you do so, your app does not persist app data from the instant app to the installed version of your app. You must set the installed app's sandbox value to 2 in order for the data to persist from the instant app to the installed version.

Once an app is installed, you can only update its target sandbox value to a higher value. To downgrade the target sandbox value, you must uninstall the app and replace it with a version whose manifest contains a lower value for this attribute.

`android:sharedUserLabel`

A user-readable label for the shared user ID. The label must be set as a reference to a string resource; it cannot be a raw string.
This attribute was introduced in API Level 3. It is meaningful only if the **`sharedUserId`** (#uid) attribute is also set.

`android:versionCode`

An internal version number. This number is used only to determine whether one version is more recent than another, with higher numbers indicating more recent versions. This is not the version number shown to users; that number is set by the **`versionName`** attribute.

The value must be set as an integer, such as "100". You can define it however you want, as long as each successive version has a higher number. For example, it could be a build number. Or you could translate a version number in "x.y" format to an integer by encoding the "x" and "y" separately in the lower and upper 16 bits. Or you could simply increase the number by one each time a new version is released.

`android:versionName`

The version number shown to users. This attribute can be set as a raw string or as a reference to a string resource. The string has no other purpose than to be displayed to users. The `versionCode` attribute holds the significant version number used internally.

`android:installLocation`

The default install location for the app.
The following keyword strings are accepted:

Value	Description
"`internalOnly`"	The app must be installed on the internal device storage only. If this is set, the app will never be installed on the external storage. If the internal storage is full, then the system will not install the app. This is also the default behavior if you do not define `android:installLocation`.
"`auto`"	The app may be installed on the external storage, but the system will install the app on the internal storage by default. If the internal storage is full, then the system will install it on the external storage. Once installed, the user can move the app to either internal or external storage through the system settings.
"`preferExternal`"	The app prefers to be installed on the external storage (SD card). There is no guarantee that the system will honor this request. The app might be installed on internal storage if the external media is unavailable or full. Once installed, the user can move the app to either internal or external storage through the system settings.

Note: By default, your app will be installed on the internal storage and cannot be installed on the external storage unless you define this attribute to be either "`auto`" or "`preferExternal`".

When an app is installed on the external storage:

- The `.apk` file is saved to the external storage, but any app data (such as databases) is still saved on the internal device memory.
- The container in which the `.apk` file is saved is encrypted with a key that allows the app to operate only on the device that installed it. (A user cannot transfer the SD card to another device and use apps installed on the card.) Though, multiple SD cards can be used with the same device.
- At the user's request, the app can be moved to the internal storage.

The user may also request to move an app from the internal storage to the external storage. However, the system will not allow the user to move the app to external storage if this attribute is set to `internalOnly`, which is the default setting.

Read App Install Location (https://developer.android.com/guide/topics/data/install-location.html) for more information about using this attribute (including how to maintain backward compatibility).

Introduced in: API Level 8.

introduced in:

API Level 1 for all attributes, unless noted otherwise in the attribute description.

see also:

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

Last updated April 17, 2018.

Twitter
Follow @AndroidDev on Twitter

Google+
Follow Android Developers on Google+

YouTube
Check out Android Developers on YouTube