Device Compatibility

In this document

See also

- Filtering on Google Play
- Providing Resources
- Android Compatibility

Android is designed to run on many different types of devices, from phones to tablets and televisions. As a developer, the range of devices provides a huge potential audience for your app. In order for your app to be successful on all these devices, it should tolerate some feature variability and provide a flexible user interface that adapts to different screen configurations.

To facilitate your effort toward that goal, Android provides a dynamic app framework in which you can provide configuration-specific app resources in static files (such as different XML layouts for different screen sizes). Android then loads the appropriate resources based on the current device configuration. So with some forethought to your app design and some additional app resources, you can publish a single application package (APK) that provides an optimized user experience on a variety of devices.

If necessary, however, you can specify your app's feature requirements and control which types of devices can install your app from Google Play Store. This page explains how you can control which devices have access to your apps, and how to prepare your apps to make sure they reach the right audience. For more information about how you can make your app adapt to different devices, read Supporting Different Devices.

What Does "Compatibility" Mean?

As you read more about Android development, you'll probably encounter the term "compatibility" in various situations. There are two types of compatibility: *device compatibility* and *app compatibility*.

Because Android is an open source project, any hardware manufacturer can build a device that runs the Android operating system. Yet, a **device is "Android compatible"** only if it can correctly run apps written for the *Android execution environment*. The exact details of the Android execution environment are defined by the Android compatibility program and each device must pass the Compatibility Test Suite (CTS) in order to be considered compatible.

As an app developer, you don't need to worry about whether a device is Android compatible, because only devices that are Android compatible include Google Play Store. So you can rest assured that users who install your app from Google Play Store are using an Android compatible device.

However, you do need to consider whether your **app is compatible** with each potential device configuration. Because Android runs on a wide range of device configurations, some features are not available on all devices. For example, some devices may not include a compass sensor. If your app's core functionality requires the use of a compass sensor, then your app is compatible only with devices that include a compass sensor.

Controlling Your App's Availability to Devices

Android supports a variety of features your app can leverage through platform APIs. Some features are hardware-based (such as a compass sensor), some are software-based (such as app widgets), and some are dependent on the platform version. Not every device supports every feature, so you may need to control your app's availability to devices based on your app's required features.

To achieve the largest user-base possible for your app, you should strive to support as many device configurations as possible using a single APK. In most situations, you can do so by disabling optional features at runtime and providing app resources with alternatives for different configurations (such as different layouts for different screen sizes). If necessary, however, you can restrict your app's availability to devices through Google Play Store based on the following device characteristics:

- Device features
- Platform version
- Screen configuration

Device features

In order for you to manage your app's availability based on device features, Android defines *feature IDs* for any hardware or software feature that may not be available on all devices. For instance, the feature ID for the compass sensor is `FEATURE_SENSOR_COMPASS` and the feature ID for app widgets is `FEATURE_APP_WIDGETS`.

If necessary, you can prevent users from installing your app when their devices don't provide a given feature by declaring it with a `<uses-feature>` element in your app's manifest file.

For example, if your app does not make sense on a device that lacks 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>
```

Google Play Store compares the features your app requires to the features available on each user's device to determine whether your app is compatible with each device. If the device does not provide all the features your app requires, the user cannot install your app.

However, if your app's primary functionality does not *require* a device feature, you should set the `required` attribute to `"false"` and check for the device feature at runtime. If the app feature is not available on the current device, gracefully degrade the corresponding app feature. For example, you can query whether a feature is available by calling `hasSystemFeature()` like this:

```
PackageManager pm = getPackageManager();
if (!pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_COMPASS)) {
    // This device does not have a compass, turn off the compass feature
    disableCompassFeature();
}
```

For information about all the filters you can use to control the availability of your app to users through Google Play Store, see the Filters on Google Play document.

> **Note:** Some system permissions implicitly require the availability of a device feature. For example, if your app requests permission to access to `BLUETOOTH`, this implicitly requires the `FEATURE_BLUETOOTH` device feature. You can disable filtering based on this feature and make your app available to devices without Bluetooth by setting the `required` attribute to `"false"` in the `<uses-feature>` tag. For more information about implicitly required device features, read Permissions that Imply Feature Requirements.

Platform version

Different devices may run different versions of the Android platform, such as Android 4.0 or Android 4.4. Each successive platform version often adds new APIs not available in the previous version. To indicate which set of APIs are available, each platform version specifies an API level. For instance, Android 1.0 is API level 1 and Android 4.4 is API level 19.

The API level allows you to declare the minimum version with which your app is compatible, using the `<uses-sdk>` manifest tag and its `minSdkVersion` attribute.

For example, the Calendar Provider APIs were added in Android 4.0 (API level 14). If your app cannot function without these APIs, you should declare API level 14 as your app's minimum supported version like this:

```
<manifest ... >
    <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="19" />
    ...
</manifest>
```

The `minSdkVersion` attribute declares the minimum version with which your app is compatible and the `targetSdkVersion` attribute declares the highest version on which you've optimized your app.

Each successive version of Android provides compatibility for apps that were built using the APIs from previous platform versions, so your app should always be compatible with future versions of Android while using the documented Android APIs.

> **Note:** The `targetSdkVersion` attribute does not prevent your app from being installed on platform versions that are higher than the specified value, but it is important because it indicates to the system whether your app should inherit behavior changes in newer versions. If you don't update the `targetSdkVersion` to the latest version, the system assumes that your app requires some backward-compatibility behaviors when running on the latest version. For example, among the behavior changes in Android 4.4, alarms created with the `AlarmManager` APIs are now inexact by default so the system can batch app alarms and preserve system power, but the system will retain the previous API behavior for your app if your target API level is lower than "19".

However, if your app uses APIs added in a more recent platform version, but does not require them for its primary functionality, you should check the API level at runtime and gracefully degrade the corresponding features when the API level is too low. In this case, set the `minSdkVersion` to the lowest value possible for your app's primary functionality, then compare the current system's version, `SDK_INT`, to one the codename constants in `Build.VERSION_CODES` that corresponds to the API level you want to check. For example:

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
    // Running on something older than API level 11, so disable
    // the drag/drop features that use ClipboardManager APIs
    disableDragAndDrop();
}
```

Screen configuration

Android runs on devices of various sizes, from phones to tablets and TVs. In order to categorize devices by their screen type, Android defines two characteristics for each device: screen size (the physical size of the screen) and screen density (the physical density of the pixels on the screen, known as DPI). To simplify the different configurations, Android generalizes these variants into groups that make them easier to target:

- Four generalized sizes: small, normal, large, and xlarge.
- And several generalized densities: mdpi (medium), hdpi (hdpi), xhdpi (extra high), xxhdpi (extra-extra high), and others.

By default, your app is compatible with all screen sizes and densities, because the system makes the appropriate adjustments to your UI layout and image resources as necessary for each screen. However, you should optimize the user experience for each screen configuration by adding specialized layouts for different screen sizes and optimized bitmap images for common screen densities.

For information about how to create alternative resources for different screens and how to restrict your app to certain screen sizes when necessary, read Supporting Different Screens.

Controlling Your App's Availability for Business Reasons

In addition to restricting your app's availability based on device characteristics, it's possible you may need to restrict your app's availability for business or legal reasons. For instance, an app that displays train schedules for the London Underground is unlikely to be useful to users outside the United Kingdom. For this type of situation, Google Play Store provides filtering options in the Play Console that allow you to control your app's availability for non-technical reasons such as the user's locale or wireless carrier.

Filtering for technical compatibility (such as required hardware components) is always based on information contained within your APK file. But filtering for non-technical reasons (such as geographic locale) is always handled in the Google Play Console.

CONTINUE READING ABOUT:

Providing Resources

Information about how Android apps are structured to separate app resources from the app code, including how you can provide alternative resources for specific device configurations.

Filters on Google Play

Information about the different ways that Google Play Store can prevent your app from being installed on different devices.

YOU MIGHT ALSO BE INTERESTED IN:

System Permissions

How Android restricts app access to certain APIs with a permission system that requires the user's consent for your app to use those APIs.