Permissions overview

The purpose of a *permission* is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

This page provides an overview to how Android permissions work, including: how permissions are presented to the user, the difference between install-time and runtime permission requests, how permissions are enforced, and the types of permissions and their groups. If you just want a how-to guide for using app permissions, instead see Request App Permissions (https://developer.android.com/training/permissions/requesting.html).

Permission approval

An app must publicize the permissions it requires by including **<uses-permission>** (https://developer.android.com/guide/topics/manifest/uses-permission-element.html) tags in the app manifest (https://developer.android.com/guide/topics/manifest/manifest-intro.html). For example, an app that needs to send SMS messages would have this line in the manifest:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.snazzyapp">

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

    <application ...>
        ...
    </application>
</manifest>
```

If your app lists *normal* permissions in its manifest (that is, permissions that don't pose much risk to the user's privacy or the device's operation), the system automatically grants those permissions to your app.

If your app lists *dangerous* permissions in its manifest (that is, permissions that could potentially affect the user's privacy or the device's normal operation), such as the **SEND_SMS** (https://developer.android.com/reference/android/Manifest.permission.html#SEND_SMS) permission above, the user must explicitly agree to grant those permissions.

For more information about normal and dangerous permissions, see Protection levels (#normal-dangerous).

Request prompts for dangerous permissions

Only dangerous permissions require user agreement. The way Android asks the user to grant dangerous permissions depends on the version of Android running on the user's device, and the system version targeted by your app.

Runtime requests (Android 6.0 and higher)

If the device is running Android 6.0 (API level 23) or higher, *and* the app's `targetSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#target) is 23 or higher, the user isn't notified of any app permissions at install time. Your app must ask the user to grant the dangerous permissions at runtime. When your app requests permission, the user sees a system dialog (as shown in figure 1, left) telling the user which permission group your app is trying to access. The dialog includes a **Deny** and **Allow** button.

If the user denies the permission request, the next time your app requests the permission, the dialog contains a checkbox that, when checked, indicates the user doesn't want to be prompted for the permission again (see figure 2, right).

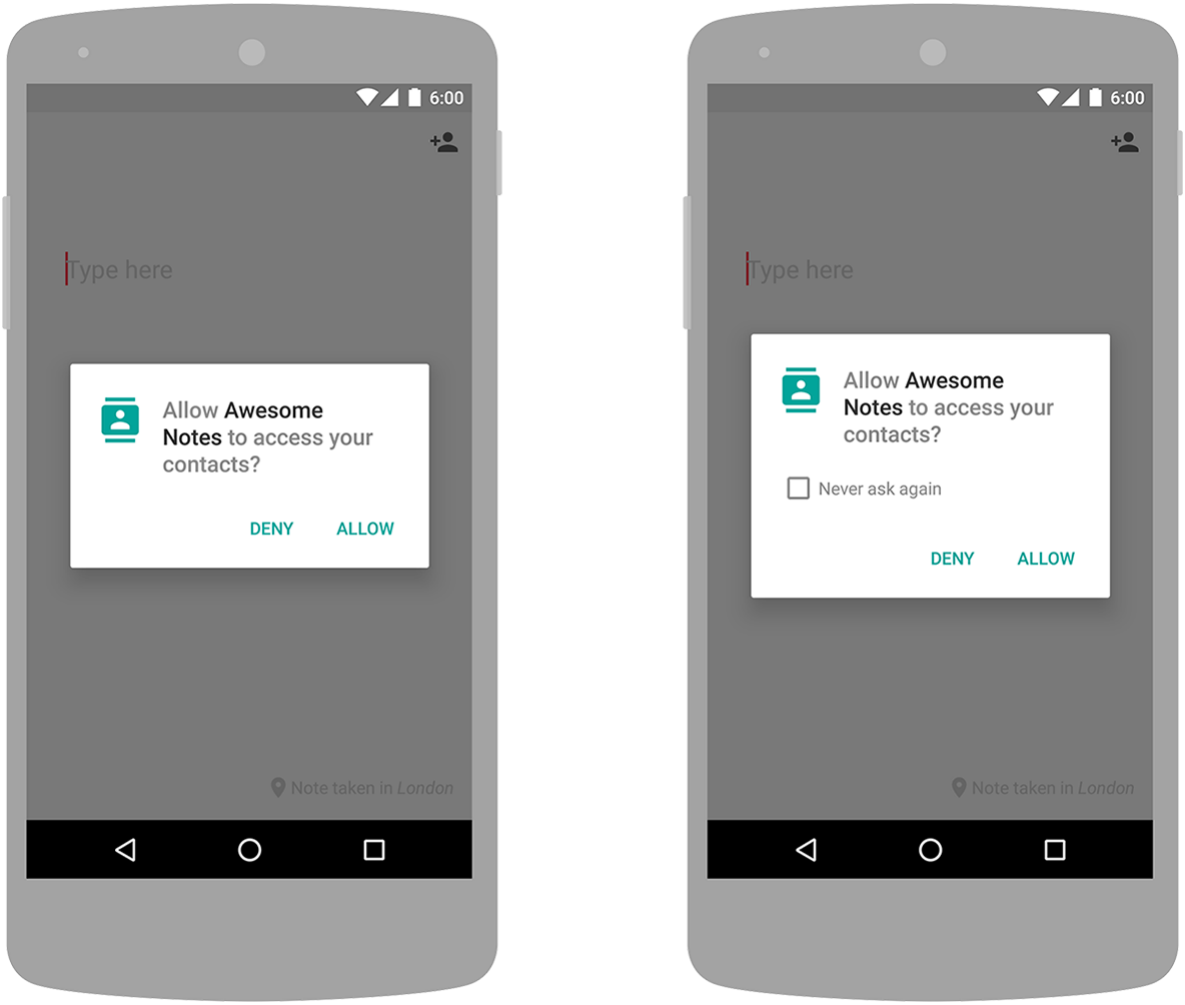


Figure 1. Initial permission dialog (left) and secondary permission request with option to turn off further requests (right)

If the user checks the **Never ask again** box and taps **Deny**, the system no longer prompts the user if you later attempt to requests the same permission.

Even if the user grants your app the permission it requested you cannot always rely on having it. Users also have the option to enable and disable permissions one-by-one in system settings. You should always check for and request permissions at runtime to guard against runtime errors (`SecurityException` (https://developer.android.com/reference/java/lang/SecurityException.html)).

For details about how to handle runtime permission requests, see Request App Permissions (https://developer.android.com/training/permissions/requesting.html).

Install-time requests (Android 5.1.1 and below)

If the device is running Android 5.1.1 (API level 22) or lower, *or* the app's `targetSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#target) is 22 or lower

while running on any version of Android, the system automatically asks the user to grant all dangerous permissions for your app at install-time (see figure 2).

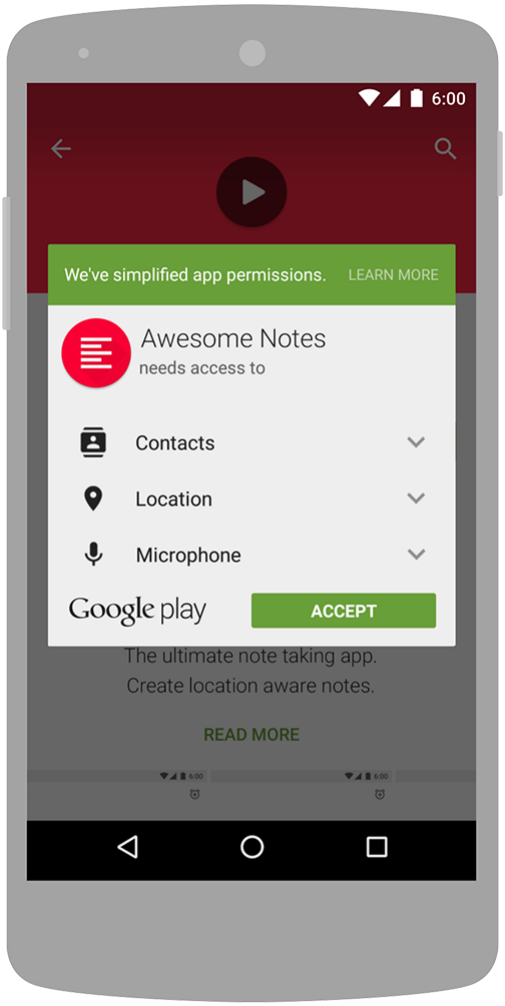


Figure 2. Install-time permission dialog

If the user clicks **Accept**, all permissions the app requests are granted. If the user denies the permissions request, the system cancels the installation of the app.

If an app update includes the need for additional permissions the user is prompted to accept those new permissions before updating the app.

For an overview of the recommended user experience patterns for requesting permissions, see App Permissions Best Practices (https://developer.android.com/training/permissions/usage-notes.html).

To learn how to check for and request permissions from the user, see Request App Permissions (https://developer.android.com/training/permissions/requesting.html).

Permissions for optional hardware features

Access to some hardware features (such as Bluetooth or the camera) require an app permission. However, not all Android devices actually have these hardware features. So if your app requests the `CAMERA` (https://developer.android.com/reference/android/Manifest.permission.html#CAMERA) permission, it's important that you also include the `<uses-feature>` (https://developer.android.com/guide/topics/manifest/uses-feature-element.html) tag in your manifest to declare whether or not this feature is actually required. For example:

```
<uses-feature android:name="android.hardware.camera" android:required="f
```

If you declare `android:required="false"` for the feature, then Google Play allows your app to be installed on devices that don't have the feature. You then must check if the current device has the feature at runtime by calling `PackageManager.hasSystemFeature()` (https://developer.android.com/reference/android/content/pm/PackageManager.html#hasSystemFeature(java.lang.String)), and gracefully disable that feature if it's not available.

If you don't provide the `<uses-feature>` (https://developer.android.com/guide/topics/manifest/uses-feature-element.html) tag, then when Google Play sees that your app requests the corresponding permission, it assumes your app requires this feature. So it filters your app from devices without the feature, as if you declared `android:required="true"` in the `<uses-feature>` (https://developer.android.com/guide/topics/manifest/uses-feature-element.html) tag.

For more information, see Google Play and feature-based filtering (https://developer.android.com/guide/topics/manifest/uses-feature-element.html#market-feature-filtering)

Permission enforcement

Permissions aren't only for requesting system functionality. Services provided by apps can enforce custom permissions to restrict who can use them. For more information on declaring custom permissions, see Define a Custom App Permission (https://developer.android.com/guide/topics/permissions/defining.html).

Activity permission enforcement

Permissions applied using the `android:permission` attribute to the `<activity>` (https://developer.android.com/guide/topics/manifest/activity-element.html) tag in the manifest restrict who can start that `Activity`

(https://developer.android.com/reference/android/app/Activity.html). The permission is checked during `Context.startActivity()` (https://developer.android.com/reference/android/content/Context.html#startActivity(android.content.Intent)) and `Activity.startActivityForResult()` (https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent, int)). If the caller doesn't have the required permission then `SecurityException` (https://developer.android.com/reference/java/lang/SecurityException.html) is thrown from the call.

Service permission enforcement

Permissions applied using the `android:permission` attribute to the `<service>` (https://developer.android.com/guide/topics/manifest/service-element.html) tag in the manifest restrict who can start or bind to the associated `Service` (https://developer.android.com/reference/android/app/Service.html). The permission is checked during `Context.startService()` (https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)), `Context.stopService()` (https://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)) and `Context.bindService()` (https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). If the caller doesn't have the required permission then `SecurityException` (https://developer.android.com/reference/java/lang/SecurityException.html) is thrown from the call.

Broadcast permission enforcement

Permissions applied using the `android:permission` attribute to the `<receiver>` (https://developer.android.com/guide/topics/manifest/receiver-element.html) tag restrict who can send broadcasts to the associated `BroadcastReceiver` (https://developer.android.com/reference/android/content/BroadcastReceiver.html). The permission is checked *after* `Context.sendBroadcast()` (https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)) returns, as the system tries to deliver the submitted broadcast to the given receiver. As a result, a permission failure doesn't result in an exception being thrown back to the caller; it just doesn't deliver the `Intent` (https://developer.android.com/reference/android/content/Intent.html).

In the same way, a permission can be supplied to `Context.registerReceiver()` (https://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter, java.lang.String, android.os.Handler)) to control who can broadcast to a programmatically registered receiver. Going the other way, a permission can be supplied when calling `Context.sendBroadcast()` (https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent, java.lang.String)) to restrict which broadcast receivers are allowed to receive the broadcast.

Note that both a receiver and a broadcaster can require a permission. When this happens, both permission checks must pass for the intent to be delivered to the associated target. For more information, see Restricting broadcasts with permissions (https://developer.android.com/guide/components/broadcasts.html#restricting_broadcasts_with_permissions) .

Content Provider permission enforcement

Permissions applied using the `android:permission` attribute to the `<provider>` (https://developer.android.com/guide/topics/manifest/provider-element.html) tag restrict who can access the data in a `ContentProvider` (https://developer.android.com/reference/android/content/ContentProvider.html). (Content providers have an important additional security facility available to them called URI permissions (#uri) which is described next.) Unlike the other components, there are two separate permission attributes you can set: `android:readPermission` (https://developer.android.com/guide/topics/manifest/provider-element.html#rprmsn) restricts who can read from the provider, and `android:writePermission` (https://developer.android.com/guide/topics/manifest/provider-element.html#wprmsn) restricts who can write to it. Note that if a provider is protected with both a read and write permission, holding only the write permission doesn't mean you can read from a provider.

The permissions are checked when you first retrieve a provider (if you don't have either permission, a `SecurityException` (https://developer.android.com/reference/java/lang/SecurityException.html) is thrown), and as you perform operations on the provider. Using `ContentResolver.query()` (https://developer.android.com/reference/android/content/ContentResolver.html#query(android.net.Uri, java.lang.String[], android.os.Bundle, android.os.CancellationSignal)) requires holding the read permission; using `ContentResolver.insert()` (https://developer.android.com/reference/android/content/ContentResolver.html#insert(android.net.Uri, android.content.ContentValues)) , `ContentResolver.update()`

(https://developer.android.com/reference/android/content/ContentResolver.html#update(android.net.Uri, android.content.ContentValues, java.lang.String, java.lang.String[]))
, `ContentResolver.delete()`
(https://developer.android.com/reference/android/content/ContentResolver.html#delete(android.net.Uri, java.lang.String, java.lang.String[]))
requires the write permission. In all of these cases, not holding the required permission results in a `SecurityException`
(https://developer.android.com/reference/java/lang/SecurityException.html) being thrown from the call.

URI permissions

The standard permission system described so far is often not sufficient when used with content providers (https://developer.android.com/guide/topics/providers/content-providers.html). A content provider may want to protect itself with read and write permissions, while its direct clients also need to hand specific URIs to other apps for them to operate on.

A typical example is attachments in a email app. Access to the emails should be protected by permissions, since this is sensitive user data. However, if a URI to an image attachment is given to an image viewer, that image viewer no longer has permission to open the attachment since it has no reason to hold a permission to access all email.

The solution to this problem is per-URI permissions: when starting an activity or returning a result to an activity, the caller can set `Intent.FLAG_GRANT_READ_URI_PERMISSION`
(https://developer.android.com/reference/android/content/Intent.html#FLAG_GRANT_READ_URI_PERMISSION)
and/or `Intent.FLAG_GRANT_WRITE_URI_PERMISSION`
(https://developer.android.com/reference/android/content/Intent.html#FLAG_GRANT_WRITE_URI_PERMISSION)
. This grants the receiving activity permission access the specific data URI in the intent, regardless of whether it has any permission to access data in the content provider corresponding to the intent.

This mechanism allows a common capability-style model where user interaction (such as opening an attachment or selecting a contact from a list) drives ad-hoc granting of fine-grained permission. This can be a key facility for reducing the permissions needed by apps to only those directly related to their behavior.

To build the most secure implementation that makes other apps accountable for their actions within yor app, you should use fine-grained permissions in this manner and declare your app's support for it with the `android:grantUriPermissions`
(https://developer.android.com/guide/topics/manifest/provider-element.html#gprmsn) attribute or

`<grant-uri-permissions>`
(https://developer.android.com/reference/android/R.styleable.html#AndroidManifestGrantUriPermission)
tag.

More information can be found in the `Context.grantUriPermission()`
(https://developer.android.com/reference/android/content/Context.html#grantUriPermission(java.lang.String, android.net.Uri, int))
, `Context.revokeUriPermission()`
(https://developer.android.com/reference/android/content/Context.html#revokeUriPermission(android.net.Uri, int))
, and `Context.checkUriPermission()`
(https://developer.android.com/reference/android/content/Context.html#checkUriPermission(android.net.Uri, int, int, int))
methods.

Other permission enforcement

Arbitrarily fine-grained permissions can be enforced at any call into a service. This is accomplished with the `Context.checkCallingPermission()`
(https://developer.android.com/reference/android/content/Context.html#checkCallingPermission(java.lang.String))
method. Call with a desired permission string and it returns an integer indicating whether that permission has been granted to the current calling process. Note that this can only be used when you are executing a call coming in from another process, usually through an IDL interface published from a service or in some other way given to another process.

There are a number of other useful ways to check permissions. If you have the process ID (PID) of another process, you can use the `Context.checkPermission()`
(https://developer.android.com/reference/android/content/Context.html#checkPermission(java.lang.String, int, int))
method to check a permission against that PID. If you have the package name of another app, you can use the `PackageManager.checkPermission()`
(https://developer.android.com/reference/android/content/pm/PackageManager.html#checkPermission(java.lang.String, java.lang.String))
method to find out whether that particular package has been granted a specific permission.

Automatic permission adjustments

Over time, new restrictions may be added to the platform such that, in order to use certain APIs, your app must request a permission that it previously did not need. Because existing

apps assume access to those APIs is freely available, Android may apply the new permission request to the app's manifest to avoid breaking the app on the new platform version (thereby, "grandfathering" your app for the permission). Android makes the decision as to whether an app might need the permission based on the value provided for the `targetSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#target) attribute. If the value is lower than the version in which the permission was added, then Android adds the permission.

For example, the `READ_EXTERNAL_STORAGE` (https://developer.android.com/reference/android/Manifest.permission.html#READ_EXTERNAL_STORAGE) permission is enforced beginning with API level 19 to restrict access to the shared storage space. If your `targetSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#target) is 18 or lower, this permission is added to your app on newer versions of Android.

Caution: If a permission is automatically added to your app, your app listing on Google Play lists these additional permissions even though your app might not actually require them. To avoid this and remove the default permissions you don't need, always update your `targetSdkVersion` to be as high as possible. You can see which permissions were added with each release in the `Build.VERSION_CODES` (https://developer.android.com/reference/android/os/Build.VERSION_CODES.html) documentation.

Protection levels

Permissions are divided into several protection levels. The protection level affects whether runtime permission requests are required.

There are three protection levels that affect third-party apps: *normal*, *signature*, and *dangerous* permissions.

Normal permissions

Normal permissions cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps. For example, permission to set the time zone is a normal permission.

If an app declares in its manifest that it needs a normal permission, the system automatically grants the app that permission at install time. The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

As of Android 8.1 (API level 27), the following permissions are classified as `PROTECTION_NORMAL` (https://developer.android.com/reference/android/content/pm/PermissionInfo.html):

- `ACCESS_LOCATION_EXTRA_COMMANDS` (https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_LOCATION_EXTRA_COMMANDS)
- `ACCESS_NETWORK_STATE` (https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_NETWORK_STATE)
- `ACCESS_NOTIFICATION_POLICY` (https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_NOTIFICATION_POLICY)
- `ACCESS_WIFI_STATE` (https://developer.android.com/reference/android/Manifest.permission.html#ACCESS_WIFI_STATE)
- `BLUETOOTH` (https://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH)
- `BLUETOOTH_ADMIN` (https://developer.android.com/reference/android/Manifest.permission.html#BLUETOOTH_ADMIN)
- `BROADCAST_STICKY` (https://developer.android.com/reference/android/Manifest.permission.html#BROADCAST_STICKY)
- `CHANGE_NETWORK_STATE` (https://developer.android.com/reference/android/Manifest.permission.html#CHANGE_NETWORK_STATE)
- `CHANGE_WIFI_MULTICAST_STATE` (https://developer.android.com/reference/android/Manifest.permission.html#CHANGE_WIFI_MULTICAST_STATE)
- `CHANGE_WIFI_STATE` (https://developer.android.com/reference/android/Manifest.permission.html#CHANGE_WIFI_STATE)
- `DISABLE_KEYGUARD` (https://developer.android.com/reference/android/Manifest.permission.html#DISABLE_KEYGUARD)
- `EXPAND_STATUS_BAR` (https://developer.android.com/reference/android/Manifest.permission.html#EXPAND_STATUS_BAR)

- GET_PACKAGE_SIZE (https://developer.android.com/reference/android/Manifest.permission.html#GET_PACKAGE_SIZE)
- INSTALL_SHORTCUT (https://developer.android.com/reference/android/Manifest.permission.html#INSTALL_SHORTCUT)
- INTERNET (https://developer.android.com/reference/android/Manifest.permission.html#INTERNET)
- KILL_BACKGROUND_PROCESSES (https://developer.android.com/reference/android/Manifest.permission.html#KILL_BACKGROUND_PROCESSES)
- MANAGE_OWN_CALLS (https://developer.android.com/reference/android/Manifest.permission.html#MANAGE_OWN_CALLS)
- MODIFY_AUDIO_SETTINGS (https://developer.android.com/reference/android/Manifest.permission.html#MODIFY_AUDIO_SETTINGS)
- NFC (https://developer.android.com/reference/android/Manifest.permission.html#NFC)
- READ_SYNC_SETTINGS (https://developer.android.com/reference/android/Manifest.permission.html#READ_SYNC_SETTINGS)
- READ_SYNC_STATS (https://developer.android.com/reference/android/Manifest.permission.html#READ_SYNC_STATS)
- RECEIVE_BOOT_COMPLETED (https://developer.android.com/reference/android/Manifest.permission.html#RECEIVE_BOOT_COMPLETED)
- REORDER_TASKS (https://developer.android.com/reference/android/Manifest.permission.html#REORDER_TASKS)
- REQUEST_COMPANION_RUN_IN_BACKGROUND (https://developer.android.com/reference/android/Manifest.permission.html#REQUEST_COMPANION_RUN_IN_BACKGROUND)
- REQUEST_COMPANION_USE_DATA_IN_BACKGROUND (https://developer.android.com/reference/android/Manifest.permission.html#REQUEST_COMPANION_USE_DATA_IN_BACKGROUND)
- REQUEST_DELETE_PACKAGES (https://developer.android.com/reference/android/Manifest.permission.html#REQUEST_DELETE_PACKAGES)

- `REQUEST_IGNORE_BATTERY_OPTIMIZATIONS` (https://developer.android.com/reference/android/Manifest.permission.html#REQUEST_IGNORE_BATTERY_OPTIMIZATIONS)
- `SET_ALARM` (https://developer.android.com/reference/android/Manifest.permission.html#SET_ALARM)
- `SET_WALLPAPER` (https://developer.android.com/reference/android/Manifest.permission.html#SET_WALLPAPER)
- `SET_WALLPAPER_HINTS` (https://developer.android.com/reference/android/Manifest.permission.html#SET_WALLPAPER_HINTS)
- `TRANSMIT_IR` (https://developer.android.com/reference/android/Manifest.permission.html#TRANSMIT_IR)
- `USE_FINGERPRINT` (https://developer.android.com/reference/android/Manifest.permission.html#USE_FINGERPRINT)
- `VIBRATE` (https://developer.android.com/reference/android/Manifest.permission.html#VIBRATE)
- `WAKE_LOCK` (https://developer.android.com/reference/android/Manifest.permission.html#WAKE_LOCK)
- `WRITE_SYNC_SETTINGS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_SYNC_SETTINGS)

Signature permissions

The system grants these app permissions at install time, but only when the app that attempts to use a permission is signed by the same certificate as the app that defines the permission.

Note: Some signature permissions aren't for use by third-party apps.

As of Android 8.1 (API level 27), the following permissions that third-party apps can use are classified as `PROTECTION_SIGNATURE` (https://developer.android.com/reference/android/content/pm/PermissionInfo.html#PROTECTION_SIGNATURE)
:

- `BIND_ACCESSIBILITY_SERVICE` (https://developer.android.com/reference/android/Manifest.permission.html#BIND_ACCESSIBILITY_SERVICE)

- BIND_AUTOFILL_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_AUTOFILL_SERVICE)
- BIND_CARRIER_SERVICES (https://developer.android.com/reference/android/Manifest.permission.html#BIND_CARRIER_SERVICES)
- BIND_CHOOSER_TARGET_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_CHOOSER_TARGET_SERVICE)
- BIND_CONDITION_PROVIDER_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_CONDITION_PROVIDER_SERVICE)
- BIND_DEVICE_ADMIN (https://developer.android.com/reference/android/Manifest.permission.html#BIND_DEVICE_ADMIN)
- BIND_DREAM_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_DREAM_SERVICE)
- BIND_INCALL_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_INCALL_SERVICE)
- BIND_INPUT_METHOD (https://developer.android.com/reference/android/Manifest.permission.html#BIND_INPUT_METHOD)
- BIND_MIDI_DEVICE_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_MIDI_DEVICE_SERVICE)
- BIND_NFC_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_NFC_SERVICE)
- BIND_NOTIFICATION_LISTENER_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_NOTIFICATION_LISTENER_SERVICE)
- BIND_PRINT_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_PRINT_SERVICE)
- BIND_SCREENING_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_SCREENING_SERVICE)

- BIND_TELECOM_CONNECTION_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_TELECOM_CONNECTION_SERVICE)
- BIND_TEXT_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_TEXT_SERVICE)
- BIND_TV_INPUT (https://developer.android.com/reference/android/Manifest.permission.html#BIND_TV_INPUT)
- BIND_VISUAL_VOICEMAIL_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_VISUAL_VOICEMAIL_SERVICE)
- BIND_VOICE_INTERACTION (https://developer.android.com/reference/android/Manifest.permission.html#BIND_VOICE_INTERACTION)
- BIND_VPN_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_VPN_SERVICE)
- BIND_VR_LISTENER_SERVICE (https://developer.android.com/reference/android/Manifest.permission.html#BIND_VR_LISTENER_SERVICE)
- BIND_WALLPAPER (https://developer.android.com/reference/android/Manifest.permission.html#BIND_WALLPAPER)
- CLEAR_APP_CACHE (https://developer.android.com/reference/android/Manifest.permission.html#CLEAR_APP_CACHE)
- MANAGE_DOCUMENTS (https://developer.android.com/reference/android/Manifest.permission.html#MANAGE_DOCUMENTS)
- READ_VOICEMAIL (https://developer.android.com/reference/android/Manifest.permission.html#READ_VOICEMAIL)
- REQUEST_INSTALL_PACKAGES (https://developer.android.com/reference/android/Manifest.permission.html#REQUEST_INSTALL_PACKAGES)
- SYSTEM_ALERT_WINDOW (https://developer.android.com/reference/android/Manifest.permission.html#SYSTEM_ALERT_WINDOW)
- WRITE_SETTINGS (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_SETTINGS)

- `WRITE_VOICEMAIL` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_VOICEMAIL)

Dangerous permissions

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user approves the permission, your app cannot provide functionality that depends on that permission.

To use a dangerous permission, your app must prompt the user to grant permission at runtime. For more details about how the user is prompted, see Request prompt for dangerous permission (#dangerous-permission-prompt).

For a list of dangerous permissions, see table 1 (#permission-groups) below.

Special permissions

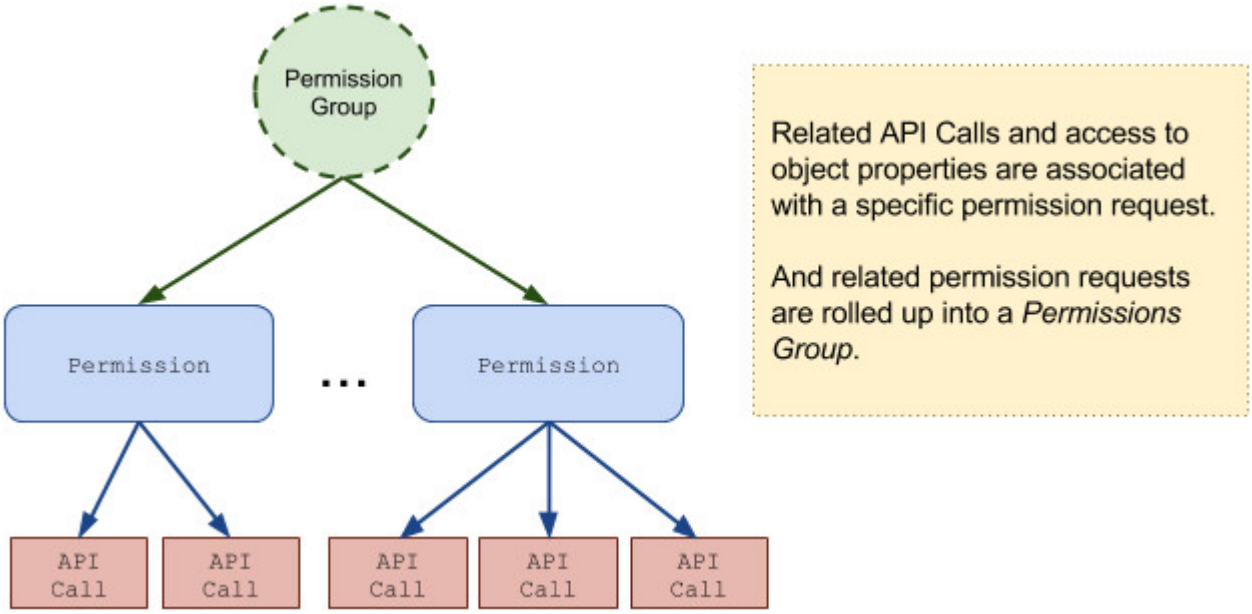
There are a couple of permissions that don't behave like normal and dangerous permissions. `SYSTEM_ALERT_WINDOW` (https://developer.android.com/reference/android/Manifest.permission.html#SYSTEM_ALERT_WINDOW) and `WRITE_SETTINGS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_SETTINGS) are particularly sensitive, so most apps should not use them. If an app needs one of these permissions, it must declare the permission in the manifest, *and* send an intent requesting the user's authorization. The system responds to the intent by showing a detailed management screen to the user.

For details on how to request these permissions, see the `SYSTEM_ALERT_WINDOW` (https://developer.android.com/reference/android/Manifest.permission.html#SYSTEM_ALERT_WINDOW) and `WRITE_SETTINGS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_SETTINGS) reference entries.

All permissions provided by the Android system can be found at `Manifest.permission` (https://developer.android.com/reference/android/Manifest.permission.html).

Permission groups

Permissions are organized into groups related to a device's capabilities or features. Under this system, permission requests are handled at the group level and a single permission group corresponds to several permission declarations in the app manifest. For example, the SMS group includes both the `READ_SMS` and the `RECEIVE_SMS` declarations. Grouping permissions in this way enables the user to make more meaningful and informed choices, without being overwhelmed by complex and technical permission requests.



All dangerous Android permissions belong to permission groups. Any permission can belong to a permission group regardless of protection level. However, a permission's group only affects the user experience if the permission is dangerous.

If the device is running Android 6.0 (API level 23) and the app's **`targetSdkVersion`** (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#target) is 23 or higher, the following system behavior applies when your app requests a dangerous permission:

- If the app doesn't currently have any permissions in the permission group, the system shows the permission request dialog to the user describing the permission group that the app wants access to. The dialog doesn't describe the specific permission within that group. For example, if an app requests the **`READ_CONTACTS`** (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission, the system dialog just says the app needs access to the device's contacts. If the user grants approval, the system gives the app just the permission it requested.
- If the app has already been granted another dangerous permission in the same permission group, the system immediately grants the permission without any

interaction with the user. For example, if an app had previously requested and been granted the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission, and it then requests `WRITE_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CONTACTS) , the system immediately grants that permission without showing the permissions dialog to the user.

Caution: Future versions of the Android SDK might move a particular permission from one group to another. Therefore, don't base your app's logic on the structure of these permission groups.

For example, `READ_CONTACTS` is in the same permission group as `WRITE_CONTACTS` as of Android 8.1 (API level 27). If your app requests the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission, and then requests the `WRITE_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#WRITE_CONTACTS) permission, don't assume that the system can automatically grant the `WRITE_CONTACTS` permission.

If the device is running Android 5.1 (API level 22) or lower, or the app's `targetSdkVersion` (https://developer.android.com/guide/topics/manifest/uses-sdk-element.html#target) is 22 or lower, the system asks the user to grant the permissions at install time. Once again, the system just tells the user what permission *groups* the app needs, not the individual permissions. For example, when an app requests `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) the install dialog lists the Contacts group. When the user accepts, only the `READ_CONTACTS` (https://developer.android.com/reference/android/Manifest.permission.html#READ_CONTACTS) permission is granted to the app.

Note: Your app still needs to explicitly request every permission it needs, even if the user has already granted another permission in the same group. In addition, the grouping of permissions into groups may change in future Android releases. Your code shouldn't have logic that depends on a set of particular permissions being in the same group.

Table 1. Dangerous permissions and permission groups.

Permission Group
`CALENDAR` (https://developer.android.com/reference/android/Manifest.permission_group.html#CALEND

CAMERA (https://developer.android.com/reference/android/Manifest.permission_group.html#CAMERA)

CONTACTS (https://developer.android.com/reference/android/Manifest.permission_group.html#CONTAC

LOCATION (https://developer.android.com/reference/android/Manifest.permission_group.html#LOCATIC

MICROPHONE (https://developer.android.com/reference/android/Manifest.permission_group.html#MICR

PHONE (https://developer.android.com/reference/android/Manifest.permission_group.html#PHONE)

SENSORS (https://developer.android.com/reference/android/Manifest.permission_group.html#SENSORS

SMS (https://developer.android.com/reference/android/Manifest.permission_group.html#SMS)

STORAGE (https://developer.android.com/reference/android/Manifest.permission_group.html#STORAGE

Viewing an app's permissions

You can view all the permissions currently defined in the system using the Settings app and the shell command **`adb shell pm list permissions`**. To use the Settings app, go to **Settings** > **Apps**. Pick an app and scroll down to see the permissions that the app uses. For developers, the adb '-s' option displays the permissions in a form similar to how the user sees them:

```
$ adb shell pm list permissions -s
All Permissions:

Network communication: view Wi-Fi state, create Bluetooth connections, full
internet access, view network state

Your location: access extra location provider commands, fine (GPS) location,
mock location sources for testing, coarse (network-based) location

Services that cost you money: send SMS messages, directly call phone numbers

...
```

You can also use the adb **`-g`** option to grant all permissions automatically when installing an app on an emulator or test device:

```
$ adb shell install -g MyApp.apk
```

Additional resources

- Request app permissions (https://developer.android.com/training/permissions/requesting.html): The how-to guide for

requesting permissions in your app.

- Permissions that imply feature requirements (https://developer.android.com/guide/topics/manifest/uses-feature-element.html#permissions): Information about how requesting some permissions implicitly restricts your app to devices that include the corresponding hardware or software feature.
- `<uses-permission>` (https://developer.android.com/guide/topics/manifest/uses-permission-element.html): API reference for the manifest tag that declares your app's required permissions.
- Device compatibility (https://developer.android.com/guide/practices/compatibility.html): Information about how Android works on different types of devices and an introduction to how you can optimize your app for each device or restrict your app's availability to different devices.
- Android Security Overview (http://source.android.com/devices/tech/security/index.html): A detailed discussion about the Android platform's security model.
- "Mother, May I?" Asking for Permissions (https://www.youtube.com/watch?v=5xVh-7ywKpE): This video from Android Dev Summit 2015 describes best practices for requesting permissions.
- Android M Permissions (https://www.youtube.com/watch?v=f17qe9vZ8RM): This video from Google I/O 2015 explains changes made to the permissions model in Android 6.0.

Next
Request app permissions →
(https://developer.android.com/training/permissions/requesting)

Last updated June 15, 2018.

Twitter
Follow @AndroidDev on Twitter

Google+
Follow Android Developers on Google+

YouTube
Check out Android Developers on YouTube