Privacy changes in Android 10

Archived: 2026-04-06 00:25:28 UTC

Android 10 (API level 29) introduces a number of features and behavior changes to better protect users' privacy. These changes extend the transparency and control that users have over their data and the capabilities they give to apps. These features might mean that specific behaviors or data that your app is depending on may behave differently compared to older versions of the platform. The impacts on your app should be minimal if your app is following current best practices for handling user data.

This page lists a summary of each change.

Top changes

This section includes the key changes in Android 10 related to privacy.

External storage access scoped to app files and media

By default, apps targeting Android 10 and higher are given scoped access into external storage, or *scoped storage*. Such apps can see the following types of files within an external storage device without needing to request any storage-related user permissions:

- Files in the app-specific directory, accessed using `getExternalFilesDir()`.
- Photos, videos, and audio clips that the app created from the media store.

To learn more about scoped storage, as well as how to share, access, and modify files that are saved on external storage devices, see the guides on how to manage files in external storage and access and modify media files.

Access to device location in the background requires permission

To support the additional control that users have over an app's access to location information, Android 10 introduces the `ACCESS_BACKGROUND_LOCATION` permission.

Unlike the `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` permissions, the `ACCESS_BACKGROUND_LOCATION` permission only affects an app's access to location when it runs in the background. An app is considered to be accessing location in the background unless one of the following conditions is satisfied:

- An activity belonging to the app is visible.
- The app is running a foreground service that has declared a foreground service type of `location`.

  To declare the foreground service type for a service in your app, set your app's `targetSdkVersion` or `compileSdkVersion` to `29` or higher. Learn more about how foreground services can continue user-initiated actions that require access to location.

If your app creates and monitors geofences and targets Android 10 (API level 29) or higher, you must declare the `ACCESS_BACKGROUND_LOCATION` permission.

Access granted automatically when targeting Android 9 or lower

If your app runs on Android 10 or higher but targets Android 9 (API level 28) or lower, the platform applies the following behavior:

- If your app declares a `<uses-permission>` element for either `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION`, the system automatically adds a `<uses-permission>` element for `ACCESS_BACKGROUND_LOCATION` during installation.
- If your app requests either `ACCESS_FINE_LOCATION` or `ACCESS_COARSE_LOCATION`, the system automatically adds `ACCESS_BACKGROUND_LOCATION` to the request.

Access when device is upgraded to Android 10

If a user grants your app access to device location – either `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION` – then upgrades their device from Android 9 to Android 10, the system automatically updates the set of location-based permissions granted to your app. The set of permissions that your app receives after the upgrade depends on its target SDK version and its defined permissions, as shown in the following table:

Table 1. Changes in location permission state after device upgrade to Android 10

Target platform version	Coarse or fine permission granted?	Background permission defined in manifest?	Updated default permission state
Android 10	Yes	Yes	Foreground and background access
Android 10	Yes	No	Foreground access only
Android 10	No	(Ignored by system)	No access
Android 9 or lower	Yes	Automatically added by the system at device upgrade time	Foreground and background access
Android 9 or lower	No	(Ignored by system)	No access

Note that the user can change this level of access even after the system automatically updates your app's access to device location. For example, the user might reduce your app's access to foreground only or revoke access entirely. Before attempting to access the device's location, particularly within a foreground service, your app should check whether the user still allows your app to receive this location information.

Access revoked when updating target API level on Android 10 devices

Consider the case where your app is already installed on a device that runs Android 10. If you update your app to target Android 10 in this situation, the device revokes the `ACCESS_BACKGROUND_LOCATION` permission.

For more information on how to retrieve the device's location while your app is in the background, see the guide on receiving periodic location updates.

Restrictions on starting activities from the background

Starting in Android 10, the system places restrictions on starting activities from the background. This behavior change helps minimize interruptions for the user and keeps the user more in control of what's shown on their screen. As long as your app starts activities as a direct result of user interaction, your app most likely isn't affected by these restrictions.

To learn more about the recommended alternative to starting activities from the background, see the guide on how to alert users of time-sensitive events in your app.

Identifiers and data

This section lists changes specific to working with device identifiers and data.

Removal of contacts affinity

Starting in Android 10, the platform doesn't keep track of contacts affinity information. As a result, if your app conducts a search on the user's contacts, the results aren't ordered by frequency of interaction.

The guide about `ContactsProvider` contains a notice describing the specific fields and methods that are obsolete on all devices starting in Android 10.

MAC address randomization

On devices that run Android 10 or higher, the system transmits randomized MAC addresses by default.

If your app handles an enterprise use case, the platform provides APIs for several operations related to MAC addresses:

- **Obtain randomized MAC address:** Device owner apps and profile owner apps can retrieve the randomized MAC address assigned to a specific network by calling `getRandomizedMacAddress()`.
- **Obtain actual, factory MAC address:** Device owner apps can retrieve a device's actual hardware MAC address by calling `getWifiMacAddress()`. This method is useful for tracking fleets of devices.

Restriction on access to /proc/net filesystem

On devices that run Android 10 or higher, apps cannot access `/proc/net`, which includes information about a device's network state. Apps that need access to this information, such as VPNs, should use the `NetworkStatsManager` or `ConnectivityManager` class.

Restriction on non-resettable device identifiers

Starting in Android 10, apps must have the `READ_PRIVILEGED_PHONE_STATE` privileged permission in order to access the device's non-resettable identifiers, which include both IMEI and serial number.

Affected methods include the following:

- `Build`
  - `getSerial()`
- `TelephonyManager`
  - `getImei()`
  - `getDeviceId()`
  - `getMeid()`
  - `getSimSerialNumber()`
  - `getSubscriberId()`

If your app doesn't have the permission and you try asking for information about non-resettable identifiers anyway, the platform's response varies based on target SDK version:

- If your app targets Android 10 or higher, a `SecurityException` occurs.
- If your app targets Android 9 (API level 28) or lower, the method returns `null` or placeholder data if the app has the `READ_PHONE_STATE` permission. Otherwise, a `SecurityException` occurs.

Many use cases don't need non-resettable device identifiers. For example, if your app uses non-resettable device identifiers for ad-tracking or user analytics purposes, use an Android Advertising ID for those specific use cases instead. To learn more, see best practices for unique identifiers.

Limited access to clipboard data

Unless your app is the default input method editor (IME) or is the app that currently has focus, your app cannot access clipboard data on Android 10 or higher.

Protection of USB device serial number

If your app targets Android 10 or higher, your app cannot read the serial number until the user has granted your app permission to access the USB device or accessory.

To learn more about working with USB devices, see the guide on how to configure USB hosts.

Camera and connectivity

This section lists changes specific to camera metadata and connectivity APIs.

Restriction on access to camera details and metadata

Android 10 changes the breadth of information that the `getCameraCharacteristics()` method returns by default. In particular, your app must have the `CAMERA` permission in order to access potentially device-specific metadata that is included in this method's return value.

To learn more about these changes, see the section about camera fields that require permission.

Restriction on enabling and disabling Wi-Fi

Apps targeting Android 10 or higher cannot enable or disable Wi-Fi. The `WifiManager.setWifiEnabled()` method always returns `false`.

If you need to prompt users to enable and disable Wi-Fi, use a settings panel.

Restrictions on direct access to configured Wi-Fi networks

To protect user privacy, manual configuration of the list of Wi-Fi networks is restricted to system apps and device policy controllers (DPCs). A given DPC can be either the device owner or the profile owner.

If your app targets Android 10 or higher, and it isn't a system app or a DPC, then the following methods don't return useful data:

- The `getConfiguredNetworks()` method always returns an empty list.
- Each network operation method that returns an integer value— `addNetwork()` and `updateNetwork()` —always returns -1.
- Each network operation that returns a boolean value— `removeNetwork()`, `reassociate()`, `enableNetwork()`, `disableNetwork()`, `reconnect()`, and `disconnect()` —always returns `false`.

If your app needs to connect to Wi-Fi networks, use the following alternative methods:

- To trigger an instant local connection to a Wi-Fi network, use `WifiNetworkSpecifier` in a standard `NetworkRequest` object.
- To add Wi-Fi networks for consideration for providing internet access to the user, work with `WifiNetworkSuggestion` objects. You can add and remove networks that appear in the auto-connect network selection dialog by calling `addNetworkSuggestions()` and `removeNetworkSuggestions()`, respectively. These methods don't require any location permissions.

Some telephony, Bluetooth, Wi-Fi APIs require FINE location permission

If your app targets Android 10 or higher, it must have the `ACCESS_FINE_LOCATION` permission in order to use several methods within the Wi-Fi, Wi-Fi Aware, or Bluetooth APIs. The following sections list the affected classes and methods.

Telephony

- `TelephonyManager`
  - `getCellLocation()`
  - `getAllCellInfo()`
  - `requestNetworkScan()`
  - `requestCellInfoUpdate()`

    - `getAvailableNetworks()`
    - `getServiceState()`
- `TelephonyScanManager`
    - `requestNetworkScan()`
- `TelephonyScanManager.NetworkScanCallback`
    - `onResults()`
- `PhoneStateListener`
    - `onCellLocationChanged()`
    - `onCellInfoChanged()`
    - `onServiceStateChanged()`

Wi-Fi

- `WifiManager`
    - `startScan()`
    - `getScanResults()`
    - `getConnectionInfo()`
    - `getConfiguredNetworks()`
- `WifiAwareManager`
- `WifiP2pManager`
- `WifiRttManager`

Bluetooth

- `BluetoothAdapter`
    - `startDiscovery()`
    - `startLeScan()`
- `BluetoothAdapter.LeScanCallback`
- `BluetoothLeScanner`
    - `startScan()`

Permissions

This section describes updates to the Android permissions model.

Restricted access to screen contents

To protect users' screen contents, Android 10 prevents silent access to the device's screen contents by changing the scope of the `READ_FRAME_BUFFER`, `CAPTURE_VIDEO_OUTPUT`, and `CAPTURE_SECURE_VIDEO_OUTPUT` permissions. As of Android 10, these permissions are signature-access only.

Apps that need to access the device's screen contents should use the `MediaProjection` API, which displays a prompt asking the user to provide consent.

User-facing permission check on legacy apps

If your app targets Android 5.1 (API level 22) or lower, users see a permissions screen when using your app on a device that runs Android 10 or higher for the first time, as shown in Figure 1. This screen gives users the opportunity to revoke access to permissions that the system previously granted to your app at install time.

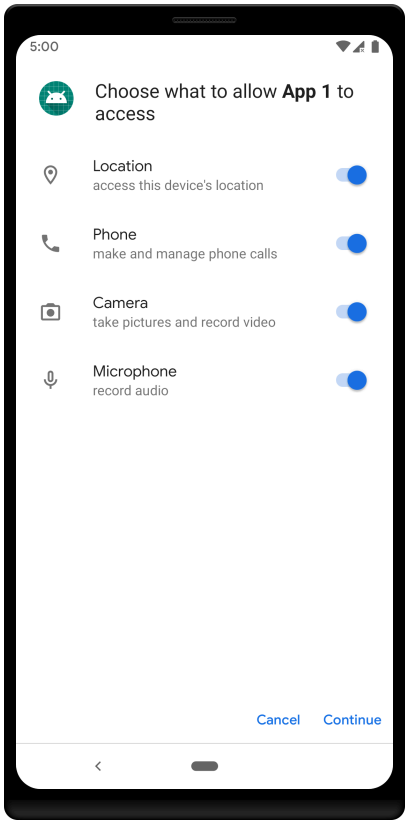

Figure 1. User-facing dialog that allows review of legacy permissions

Physical activity recognition

Android 10 introduces the `android.permission.ACTIVITY_RECOGNITION` runtime permission for apps that need to detect the user's step count or classify the user's physical activity, such as walking, biking, or moving in a vehicle. This is designed to give users visibility of how device sensor data is used in Settings.

Some libraries within Google Play services, such as the Activity Recognition API and the Google Fit API, don't provide results unless the user has granted your app this permission.

The only built-in sensors on the device that require you to declare this permission are the step counter and step detector sensors.

If your app targets Android 9 (API level 28) or lower, the system auto-grants the `android.permission.ACTIVITY_RECOGNITION` permission to your app, as needed, if your app satisfies each of the following conditions:

- The manifest file includes the `com.google.android.gms.permission.ACTIVITY_RECOGNITION` permission.
- The manifest file **doesn't** include the `android.permission.ACTIVITY_RECOGNITION` permission.

If the system-auto grants the `android.permission.ACTIVITY_RECOGNITION` permission, your app retains the permission after you update your app to target Android 10. However, the user can revoke this permission at any time in system settings.

Permission groups removed from UI

As of Android 10, apps cannot look up how permissions are grouped in the UI.
