Behavior changes: all apps

Archived: 2026-04-05 21:56:27 UTC

Android 10 includes behavior changes that may affect your app. The changes listed on this page apply to your app when running on Android 10, regardless of the app's `targetSdkVersion`. You should test your app and modify it as needed to support these changes properly.

If your app's targetSdkVersion is `29` or higher, you'll also need to support additional changes. Make sure to read behavior changes for apps targeting 29 for details.

Note: In addition to the changes listed on this page, Android 10 introduces a large number of privacy-based changes and restrictions, including the following:

- Background access to device location
- Background activity starts
- Contacts affinity information
- MAC address randomization
- Camera metadata
- Permissions model

These changes affect all apps and enhance user privacy. To learn more about how to support these changes, see the Privacy changes page.

Non-SDK interface restrictions

To help ensure app stability and compatibility, the platform started restricting which non-SDK interfaces your app can use in Android 9 (API level 28). Android 10 includes updated lists of restricted non-SDK interfaces based on collaboration with Android developers and the latest internal testing. Our goal is to make sure that public alternatives are available before we restrict non-SDK interfaces.

If you will not be targeting Android 10 (API level 29), some of these changes might not immediately affect you. However, while you can currently use some non-SDK interfaces (depending on your app's target API level), using any non-SDK method or field always carries a high risk of breaking your app.

If you are unsure if your app uses non-SDK interfaces, you can test your app to find out. If your app relies on non-SDK interfaces, you should begin planning a migration to SDK alternatives. Nevertheless, we understand that some apps have valid use cases for using non-SDK interfaces. If you cannot find an alternative to using a non-SDK interface for a feature in your app, you should request a new public API.

To learn more, see Updates to non-SDK interface restrictions in Android 10 and see Restrictions on non-SDK interfaces.

Gesture Navigation

Beginning with Android 10, users can enable gesture navigation across the device. If a user enables gesture navigation, this affects all apps on the device, whether or not the app targets API level 29. For example, if the user swipes in from the edge of the screen, the system interprets that gesture as a Back navigation, unless an app specifically overrides that gesture for portions of the screen.

To make your app compatible with gesture navigation, you'll want to extend the app content from edge to edge, and handle conflicting gestures appropriately. For information, see the Gesture navigation documentation.

NDK

Android 10 includes the following NDK changes.

Shared objects cannot contain text relocations

Android 6.0 (API level 23) disallowed use of text relocations in shared objects. Code must be loaded as-is, and must not be modified. This change improves app load times and security.

SELinux enforces this restriction on apps that target Android 10 or higher. If these apps continue to use shared objects that contain text relocations, they're at high risk of breaking.

Changes to Bionic libraries and dynamic linker paths

Starting in Android 10, several paths are symbolic links instead of regular files. Apps that have been relying on the paths being regular files might break:

- `/system/lib/libc.so` -> `/apex/com.android.runtime/lib/bionic/libc.so`
- `/system/lib/libm.so` -> `/apex/com.android.runtime/lib/bionic/libm.so`
- `/system/lib/libdl.so` -> `/apex/com.android.runtime/lib/bionic/libdl.so`
- `/system/bin/linker` -> `/apex/com.android.runtime/bin/linker`

These changes apply to the 64-bit variants of the file as well, with `lib/` replaced with `lib64/`.

For compatibility, the symlinks are provided at the old paths. For example, `/system/lib/libc.so` is a symlink to `/apex/com.android.runtime/lib/bionic/libc.so`. So `dlopen("/system/lib/libc.so")` continues to work, but apps will find the difference when they actually try to examine the loaded libraries by reading `/proc/self/maps` or similar, which is not usual but we've found that some apps do that as part of their anti-hacking process. If so, the `/apex/⋯` paths should be added as valid paths for the Bionic files.

System binaries/libraries mapped to execute-only memory

Starting in Android 10, executable segments of system binaries and libraries are mapped into memory execute-only (non-readable) as a hardening technique against code-reuse attacks. If your app performs read operations into memory segments marked as execute-only – whether from bug, vulnerability, or intentional memory inspection – the system sends a `SIGSEGV` signal to your app.

You can identify whether this behavior caused a crash by examining the related tombstone file in `/data/tombstones/`. An execute-only related crash contains the following abort message:

```
Cause: execute-only (no-read) memory access error; likely due to data in .text.
```

To work around this issue to perform operations like memory inspection, it's possible to mark execute-only segments as read+execute by calling `mprotect()`. However, we strongly recommend setting it back to execute-only afterwards, as this access permission setting provides better protection for your app and users.

Security

Android 10 includes the following security changes.

TLS 1.3 enabled by default

In Android 10 and higher, TLS 1.3 is enabled by default for all TLS connections. Here are a few important details about our TLS 1.3 implementation:

- The TLS 1.3 cipher suites cannot be customized. The supported TLS 1.3 cipher suites are always enabled when TLS 1.3 is enabled. Any attempt to disable them by calling `setEnabledCipherSuites()` is ignored.
- When TLS 1.3 is negotiated, `HandshakeCompletedListener` objects are called *before* sessions are added to the session cache. (In TLS 1.2 and other previous versions, these objects are called *after* sessions are added to the session cache.)
- In some situations where `SSLEngine` instances throw an `SSLHandshakeException` on previous versions of Android, these instances throw an `SSLProtocolException` instead on Android 10 and higher.
- 0-RTT mode isn't supported.

If desired, you can obtain an `SSLContext` that has TLS 1.3 disabled by calling `SSLContext.getInstance("TLSv1.2")`. You can also enable or disable protocol versions on a per-connection basis by calling `setEnabledProtocols()` on an appropriate object.

Certificates signed with SHA-1 aren't trusted in TLS

In Android 10, certificates that use the SHA-1 hash algorithm aren't trusted in TLS connections. Root CAs haven't issued such certificate since 2016, and they are no longer trusted in Chrome or other major browsers.

Any attempt to connect fails if the connection is to a site that presents a certificate using SHA-1.

KeyChain behavior changes and improvements

Some browsers, such as Google Chrome, allow users to choose a certificate when a TLS server sends a certificate request message as part of a TLS handshake. As of Android 10, `KeyChain` objects honor the issuers and key specification parameters when calling `KeyChain.choosePrivateKeyAlias()` to show users a certificate selection prompt. In particular, this prompt doesn't contain choices that don't adhere to server specifications.

If there are no user-selectable certificates available, as is the case when no certificates match the server specification or a device doesn't have any certificates installed, the certificate selection prompt doesn't appear at all.

In addition, it isn't necessary on Android 10 or higher to have a device screen lock to import keys or CA certificates into a `KeyChain` object.

Other TLS and cryptography changes

There have been several minor changes in the TLS and cryptography libraries that take effect on Android 10:

- The AES/GCM/NoPadding and ChaCha20/Poly1305/NoPadding ciphers return more accurate buffer sizes from `getOutputSize()`.
- The `TLS_FALLBACK_SCSV` cipher suite is omitted from connection attempts with a max protocol of TLS 1.2 or above. Because of improvements in TLS server implementations, we don't recommend attempting TLS-external fallback. Instead, we recommend relying on TLS version negotiation.
- ChaCha20-Poly1305 is an alias for ChaCha20/Poly1305/NoPadding.
- Hostnames with trailing dots aren't considered valid SNI hostnames.
- The supported_signature_algorithms extension in `CertificateRequest` is respected when choosing a signing key for certificate responses.
- Opaque signing keys, such as those from Android Keystore, can be used with RSA-PSS signatures in TLS.

Wi-Fi Direct broadcasts

On Android 10, the following broadcasts related to Wi-Fi Direct aren't sticky:

- `WIFI_P2P_CONNECTION_CHANGED_ACTION`
- `WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`

If your app has relied on receiving these broadcasts at registration because they had been sticky, use the appropriate `get()` method at initialization to obtain the information instead.

Wi-Fi Aware capabilities

Android 10 adds support to ease TCP/UDP Socket creation using Wi-Fi Aware datapaths. To create a TCP/UDP socket connecting to a `ServerSocket`, the client device needs to know the IPv6 address and port of the server. This previously needed to be communicated out-of-band, such as by using BT or Wi-Fi Aware layer 2 messaging, or discovered in-band using other protocols, such as mDNS. With Android 10, the information can be communicated as part of the network setup.

The server can do either of the following:

- Initialize a `ServerSocket` and either set or obtain the port to be used.
- Specify the port information as part of the Wi-Fi Aware network request.

The following code sample shows how to specify port information as part of the network request:

```
val ss = ServerSocket()
val ns = WifiAwareNetworkSpecifier.Builder(discoverySession, peerHandle)
  .setPskPassphrase("some-password")
  .setPort(ss.localPort)
  .build()

val myNetworkRequest = NetworkRequest.Builder()
  .addTransportType(NetworkCapabilities.TRANSPORT_WIFI_AWARE)
  .setNetworkSpecifier(ns)
  .build()
```

```
ServerSocket ss = new ServerSocket();
WifiAwareNetworkSpecifier ns = new WifiAwareNetworkSpecifier
  .Builder(discoverySession, peerHandle)
  .setPskPassphrase(“some-password”)
  .setPort(ss.getLocalPort())
  .build();

NetworkRequest myNetworkRequest = new NetworkRequest.Builder()
  .addTransportType(NetworkCapabilities.TRANSPORT_WIFI_AWARE)
  .setNetworkSpecifier(ns)
  .build();
```

The client then performs a Wi-Fi Aware network request to obtain the IPv6 and the port supplied by the server:

```
val callback = object : ConnectivityManager.NetworkCallback() {
  override fun onAvailable(network: Network) {
    ...
  }

  override fun onLinkPropertiesChanged(network: Network,
      linkProperties: LinkProperties) {
    ...
  }

  override fun onCapabilitiesChanged(network: Network,
      networkCapabilities: NetworkCapabilities) {
    ...
    val ti = networkCapabilities.transportInfo
    if (ti is WifiAwareNetworkInfo) {
       val peerAddress = ti.peerIpv6Addr
       val peerPort = ti.port
    }
  }
  override fun onLost(network: Network) {
```

```
    ...
  }
};

connMgr.requestNetwork(networkRequest, callback)
```

```
callback = new ConnectivityManager.NetworkCallback() {
  @Override
  public void onAvailable(Network network) {
    ...
  }
  @Override
  public void onLinkPropertiesChanged(Network network,
      LinkProperties linkProperties) {
    ...
  }
  @Override
  public void onCapabilitiesChanged(Network network,
      NetworkCapabilities networkCapabilities) {
    ...
    TransportInfo ti = networkCapabilities.getTransportInfo();
    if (ti instanceof WifiAwareNetworkInfo) {
       WifiAwareNetworkInfo info = (WifiAwareNetworkInfo) ti;
       Inet6Address peerAddress = info.getPeerIpv6Addr();
       int peerPort = info.getPort();
    }
  }
  @Override
  public void onLost(Network network) {
    ...
  }
};

connMgr.requestNetwork(networkRequest, callback);
```

SYSTEM_ALERT_WINDOW on Go devices

Apps running on Android 10 (Go edition) devices cannot receive the `SYSTEM_ALERT_WINDOW` permission. This is because drawing overlay windows uses excessive memory, which is particularly harmful to the performance of low-memory Android devices.

If an app running on a Go edition device running Android 9 or lower receives the `SYSTEM_ALERT_WINDOW` permission, the app retains the permission even if the device is upgraded to Android 10. However, apps which do not already have that permission cannot be granted it after the device is upgraded.

If an app on a Go device sends an intent with the action `ACTION_MANAGE_OVERLAY_PERMISSION`, the system automatically denies the request, and takes the user to a **Settings** screen which says that the permission isn't allowed because it slows the device. If an app on a Go device calls `Settings.canDrawOverlays()`, the method always returns false. Again, these restrictions do not apply to apps which received the `SYSTEM_ALERT_WINDOW` permission before the device was upgraded to Android 10.

Warnings for apps targeting older Android versions

Devices running Android 10 or higher warn users the first time they run any app that targets Android 5.1 (API level 22) or lower. If the app requires the user to grant permissions, the user is also given an opportunity to adjust the app's permissions before the app is allowed to run for the first time.

Due to Google Play's target API requirements, a user sees these warnings only when they run an app that hasn't been updated recently. For apps that are distributed through other stores, similar target API requirements are taking effect during 2019. For more information about these requirements, see Expanding target API level requirements in 2019.

SHA-2 CBC cipher suites removed

The following SHA-2 CBC cipher suites have been removed from the platform:

- `TLS_RSA_WITH_AES_128_CBC_SHA256`
- `TLS_RSA_WITH_AES_256_CBC_SHA256`
- `TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`
- `TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`

These cipher suites are less secure than the similar cipher suites that use GCM, and most servers either support both the GCM and CBC variants of these cipher suites or support neither of them.

App usage

Android 10 introduces the following behavior changes related to app usage:

- **UsageStats app usage improvements** - Android 10 accurately tracks app usage with UsageStats when apps are used in split-screen or picture-in-picture mode. Additionally, Android 10 correctly tracks instant app usage.
- **Per-app grayscale** - Android 10 can set a grayscale display mode on a per-app basis.
- **Per-app distraction state** - Android 10 can selectively set apps to a "distraction state" where their notifications are suppressed and they do not appear as suggested apps.
- **Suspension and playback** - In Android 10, suspended apps are not able to play audio.

HTTPS connection changes

If an app running Android 10 passes `null` into `setSSLSocketFactory()`, an `IllegalArgumentException` occurs. In previous versions, passing `null` into `setSSLSocketFactory()` had the same effect as passing in the current default factory.

android.preference library is deprecated

The `android.preference` library is deprecated as of Android 10. Developers should instead use the AndroidX preference library, part of Android Jetpack. For additional resources to aid in migration and development, check out the updated Settings Guide along with our public sample app and reference documentation.

ZIP file utility library changes

Android 10 introduces the following changes to classes in the `java.util.zip` package, which handles ZIP files. These changes make the library's behavior more consistent between Android and other platforms that use `java.util.zip`.

Inflater

In previous versions, some methods in the `Inflater` class threw an `IllegalStateException` if they were invoked after a call to `end()`. In Android 10, these methods throw a `NullPointerException` instead.

ZipFile

In Android 10 and higher, the constructor for `ZipFile` that takes arguments of type `File`, `int`, and `Charset` doesn't throw a `ZipException` if the supplied ZIP file doesn't contain any files.

ZipOutputStream

In Android 10 and higher, the `finish()` method in `ZipOutputStream` doesn't throw a `ZipException` if it tries to write an output stream for a ZIP file that doesn't contain any files.

Camera changes

Many camera-using apps assume that if the device is in a portrait configuration, then the physical device is also in the portrait orientation, as described in Camera orientation. This was a safe assumption in the past, but that has changed with the expansion of available form factors, such as foldables. That assumption on these devices can lead to either incorrectly rotated or scaled (or both) display of the camera viewfinder.

Applications that target API level 24 or above should explicitly set `android:resizeableActivity` and provide necessary functionality to handle multi-window operation.

Battery usage tracking

Beginning with Android 10, `SystemHealthManager` resets its battery usage statistics whenever the device is unplugged after a *major charging event*. Broadly speaking, a major charging event is either: The device has been fully charged, or the device has gone from mostly depleted to mostly charged.

Before Android 10, the battery usage statistics reset whenever the device was unplugged, no matter how little change there had been to the battery level.

Android Beam deprecation

In Android 10 we're officially deprecating Android Beam, an older feature for initiating data sharing across devices through Near Field Communication (NFC). We're also deprecating several of the related NFC APIs. Android Beam remains optionally available to device-maker partners who want to use it, but it's no longer in active development. Android will continue to support other NFC capabilities and APIs, however, and use-cases like reading from tags and payments will continue to work as expected.

java.math.BigDecimal.stripTrailingZeros() behavior change

`BigDecimal.stripTrailingZeros()` no longer preserves trailing zeroes as a special case if the input value is zero.

java.util.regex.Matcher and Pattern behavior changes

The result of `split()` was changed to no longer start with an empty `String` ("") when there is a zero-width match at the start of the input. This also affects `String.split()`. For example, `"x".split("")` now returns `{"x"}` whereas it used to return `{"", "x"}` on older versions of Android. `"aardvark".split("(?=a)"` now returns `{"a", "ardv", "ark"}` rather than `{"", "a", "ardv", "ark"}`.

Exception behavior for invalid arguments has also been improved:

- `appendReplacement(StringBuffer, String)` now throws an `IllegalArgumentException` instead of `IndexOutOfBoundsException` if the replacement `String` ends with a lone backslash, which is illegal. The same exception is now thrown if the replacement `String` ends with a `$`. Previously, no exception was thrown in this scenario.
- `replaceFirst(null)` no longer calls `reset()` on the `Matcher` if it throws a `NullPointerException`. `NullPointerException` is now also thrown when there is no match. Previously, it was thrown only when there was a match.
- `start(int group)`, `end(int group)` and `group(int group)` now throw a more general `IndexOutOfBoundsException` if the group index is out of bounds. Previously, these methods threw `ArrayIndexOutOfBoundsException`.

Default angle for GradientDrawable is now TOP_BOTTOM

In Android 10, if you define a `GradientDrawable` in XML and do not provide an angle measurement, the gradient orientation defaults to `TOP_BOTTOM`. This is a change from previous versions of Android, where the default was `LEFT_RIGHT`.

As a workaround, if you update to the most recent version of AAPT2, the tool sets an angle measurement of 0 for legacy apps if no angle measurement is specified.

Logging for serialized objects using default SUID

Beginning with Android 7.0 (API level 24), the platform made a fix to the default `serialVersionUID` for serializable objects. This fix did not affect apps that targeted API level 23 or lower.

Beginning with Android 10, if an app targets API level 23 or lower and relies on the old, incorrect, default `serialVersionUID`, the system logs a warning and suggests a code fix.

Specifically, the system logs a warning if all of the following are true:

- The app targets API level 23 or lower.
- A class is serialized.
- The serialized class uses the default `serialVersionUID`, instead of explicitly setting a `serialVersionUID`.
- The default `serialVersionUID` is different than the `serialVersionUID` would be if the app targeted API level 24 or higher.

This warning is logged once for each affected class. The warning message includes a suggested fix, which is to explicitly set the `serialVersionUID` to the default value that would be calculated if the app targeted API level 24 or higher. By using that fix, you can ensure that if an object from that class is serialized on an app that targets API level 23 or lower, the object will be correctly read by apps that target 24 or higher, and vice versa.

java.io.FileChannel.map() changes

Starting in Android 10, `FileChannel.map()` isn't supported for non-standard files, like `/dev/zero`, whose size cannot be changed using `truncate()`. Previous versions of Android swallowed the errno returned by `truncate()`, but Android 10 throws an IOException. If you need the old behavior, you must use native code.

Source: https://developer.android.com/about/versions/10/behavior-changes-all#execute-permission