Features and APIs Overview

Archived: 2026-04-05 14:58:50 UTC

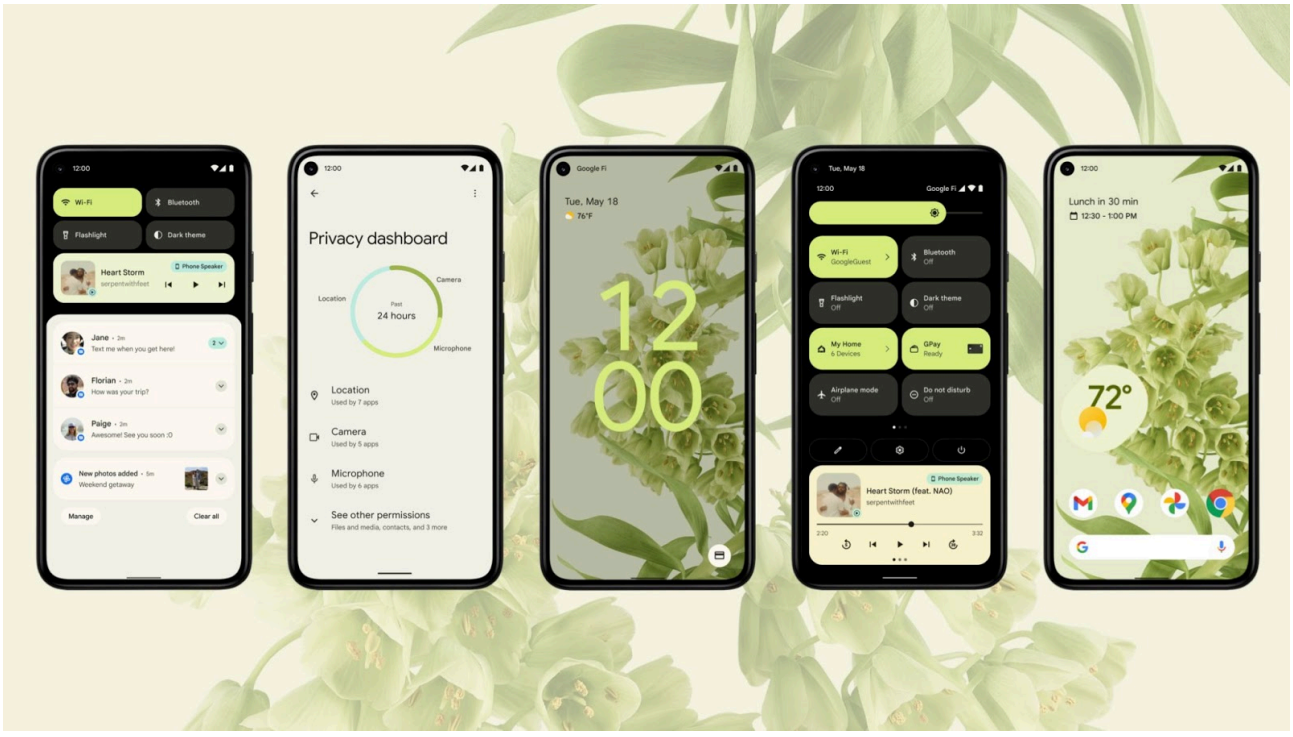
Android 12 introduces great new features and APIs for developers. The sections below help you learn about features for your apps and get started with the related APIs.

For a detailed list of new, modified, and removed APIs, read the API diff report. For details on new APIs visit the Android API reference — new APIs are highlighted for visibility. Also, to learn about areas where platform changes may affect your apps, be sure to check out Android 12 behavior changes for apps that target Android 12 and for all apps.

User experience

Material You

Android 12 introduces a new design language called Material You, helping you to build more personalized, beautiful apps. To bring all of the latest Material Design 3 updates into your apps, try an alpha version of Material Design Components.



Widgets improvements

Android 12 revamps the existing Widgets API to improve the user and developer experience in the platform and launchers. We've created a guide to help you ensure your widget is compatible with Android 12 and to refresh it with new features.

See Android 12 widgets improvements for more information.

Rich content insertion

Android 12 introduces a new unified API that lets your app receive rich content from any available source: clipboard, keyboard, or drag and drop.

For more information, see Receive rich content.

App splash screens API

Android 12 introduces a new app launch animation for all apps that includes an into-app motion from the point of launch, a splash screen showing the app icon, and a transition to the app itself. See the splash screens developer guide for more details.

Rounded corner APIs

Android 12 introduces `RoundedCorner` and `WindowInsets.getRoundedCorner(int position)`, which provide the radius and center point for rounded corners.

For more information, see Rounded corners.

Rich haptic experiences

Android 12 expands the tools for creating informative haptic feedback for UI events, immersive and delightful effects for gaming, and attentional haptics for productivity.

Actuator effects

Android 12 adds expressive effects like low tick that take advantage of the broader frequency bandwidth of the latest actuators. Game developers can now access multiple, different actuators independently in game controllers to deliver the same effect synchronously or different haptic effects on multiple actuators. For developers, we recommend using the constants and primitives as building blocks for rich haptic effects - constants to enhance UI events and haptic composer to sequence primitives for more complex effects. These APIs are available to try on Pixel 4 devices, and we're continuing to work with our device-maker partners to bring the latest in haptics support to users across the ecosystem.

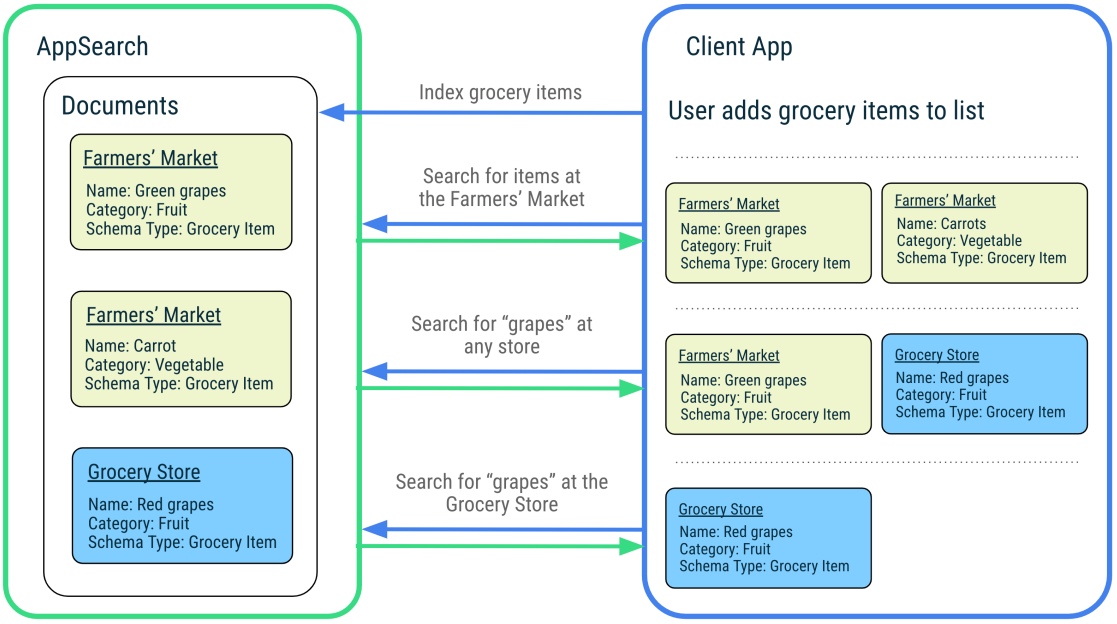
Audio-coupled haptic effects

Android 12 apps can generate haptic feedback derived from an audio session using the phone's vibrator. This provides an opportunity for more immersive game and audio experiences. For example, haptic-enhanced ringtones can help identify callers, or a driving game could simulate the feeling of rough terrain.

See the `HapticGenerator` reference documentation for more information.

AppSearch

Android 12 introduces AppSearch, a high-performance on-device search engine, as a system service. AppSearch allows applications to index structured data and search over it with built-in full-text search capabilities. Furthermore, AppSearch supports native search features, like highly-efficient indexing and retrieval, multi-language support, and relevancy ranking.



AppSearch comes in two flavors: a local index for your application to use that's compatible with older versions of Android, or a central index maintained for the entire system in Android 12. Using the central index, your application can allow its data to be displayed on system UI surfaces by the system's pre-installed intelligence component. Exactly which data gets displayed on system UI surfaces is dependent on the OEM. Additionally, your application can securely share data with other applications, to allow them to search over that data as well.

Learn more about AppSearch in the developer guide, and begin using it with the AppSearch Jetpack library, which provides a developer-friendly API surface as well as annotation processor support.

Game Mode

The Game Mode API and Game Mode interventions allow you to optimize gameplay by prioritizing characteristics, such as performance or battery life based on users settings or game specific configurations.

For more information, see Game Mode.

Picture-in-picture (PiP) recommendations and improvements

Android 12 introduces the following improvements for PiP mode:

Support for new PiP gestures

Android 12 now supports stashing and pinch-to-zoom gestures for the PiP window:

- To stash the window, the user can drag the window to the left or right edge. To unstash the window, the user can either tap the visible part of the stashed window or drag it out.
- The user can now resize the PiP window using pinch-to-zoom.

Recommended new features that support a polished PiP transition experience

Android 12 added significant cosmetic improvements to the animated transitions between fullscreen and PiP windows. We strongly recommend implementing all applicable changes; once you've done so, these changes automatically scale to large screens such as foldables and tablets without any further required work.

These features are the following:

- A new API flag for smoother transition to PiP mode with gesture navigation

  Use the `setAutoEnterEnabled` flag to provide smoother transitions to PiP mode when swiping up to home in gesture navigation mode. Previously, Android waited for the swipe-up-to-home animation to finish before fading in the PiP window.

- Smoother animations when entering and exiting out of PiP mode

  The `SourceRectHint` flag is now reused to implement smoother animation when entering and exiting PiP mode.

- A new API flag to disable seamless resizing for non-video content

  The `SeamlessResizeEnabled` flag provides a much smoother cross-fading animation when resizing non-video content in the PiP window. Previously, resizing non-video content in a PiP window could create jarring visual artifacts.

New phone call notifications allowing for ranking importance of incoming calls

Android 12 adds the new notification style `Notification.CallStyle` for phone calls. Using this template lets your app indicate the importance of active calls by displaying a prominent chip that shows the time of the call in the status bar; the user can tap this chip to return to their call.

Because incoming and ongoing calls are the most critical to users, these notifications are given top ranking in the shade. This ranking also allows the system to potentially forward these prioritized calls to other devices.

Implement the following code for all types of calls.

```
// Create a new call with the user as caller.
val incoming_caller = Person.Builder()
    .setName("Jane Doe")
    .setImportant(true)
    .build()
```

```
// Create a new call with the user as caller.
Person incoming_caller = new Person.Builder()
    .setName("Jane Doe")
    .setImportant(true)
    .build();
```

Use `forIncomingCall()` to create a call style notification for an incoming call.

```
// Create a call style notification for an incoming call.
val builder = Notification.Builder(context, CHANNEL_ID)
    .setContentIntent(contentIntent)
    .setSmallIcon(smallIcon)
    .setStyle(
         Notification.CallStyle.forIncomingCall(caller, declineIntent, answerIntent))
    .addPerson(incoming_caller)
```

```
// Create a call style notification for an incoming call.
Notification.Builder builder = Notification.Builder(context, CHANNEL_ID)
    .setContentIntent(contentIntent)
    .setSmallIcon(smallIcon)
    .setStyle(
        Notification.CallStyle.forIncomingCall(caller, declineIntent, answerIntent))
    .addPerson(incoming_caller);
```

Use `forOngoingCall()` to create a call style notification for an ongoing call.

```
// Create a call style notification for an ongoing call.
val builder = Notification.Builder(context, CHANNEL_ID)
    .setContentIntent(contentIntent)
    .setSmallIcon(smallIcon)
    .setStyle(
         Notification.CallStyle.forOnGoingCall(caller, hangupIntent))
    .addPerson(second_caller)
```

```
// Create a call style notification for an ongoing call.
Notification.Builder builder = Notification.Builder(context, CHANNEL_ID)
    .setContentIntent(contentIntent)
    .setSmallIcon(smallIcon)
    .setStyle(
        Notification.CallStyle.forOnGoingCall(caller, hangupIntent))
    .addPerson(second_caller);
```

Use `forScreeningCall()` to create a call style notification for screening a call.

```
// Create a call style notification for screening a call.
val builder = Notification.Builder(context, CHANNEL_ID)
    .setContentIntent(contentIntent)
    .setSmallIcon(smallIcon)
    .setStyle(
         Notification.CallStyle.forScreeningCall(caller, hangupIntent, answerIntent))
    .addPerson(second_caller)
```

```
Notification.Builder builder = Notification.Builder(context, CHANNEL_ID)
    .setContentIntent(contentIntent)
    .setSmallIcon(smallIcon)
    .setStyle(
        Notification.CallStyle.forScreeningCall(caller, hangupIntent, answerIntent))
    .addPerson(second_caller);
```

Enriched image support for notifications

In Android 12, you can now enrich your app's notification experience by providing animated images in `MessagingStyle()` and `BigPictureStyle()` notifications. Also, your app can now enable users to send image messages when they reply to messages from the notification shade.

Immersive mode improvements for gesture navigation

Android 12 consolidates existing behavior to make it easier for users to perform gesture navigation commands while in immersive mode. In addition, Android 12 provides backward compatibility behavior for sticky immersive mode.

Recents URL sharing (Pixel only)

On Pixel devices, users can now share links to recently viewed web content directly from the Recents screen. After visiting the content in an app, the user can swipe to the Recents screen and find the app where they viewed the content, then tap on the link button to copy or share the URL.

For more information, see Enable recents URL sharing.

Security and privacy

Privacy Dashboard

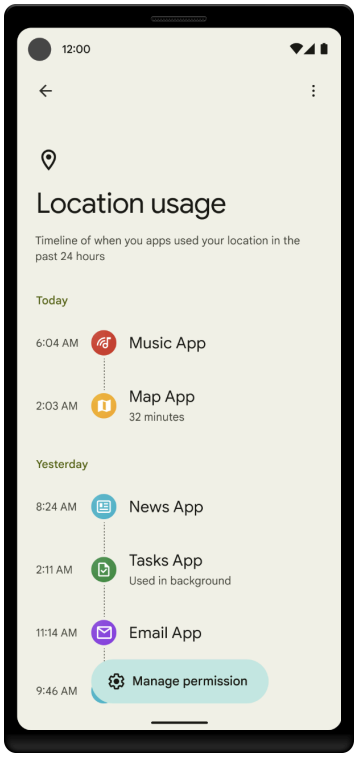


Figure 1. Location usage screen, part of the Privacy Dashboard.

On supported devices that run Android 12 or higher, a Privacy Dashboard screen appears in system settings. On this screen, users can access separate screens that show when apps access location, camera, and microphone information. Each screen shows a timeline of when different apps have accessed a particular type of data. Figure 1 shows the data access timeline for location information.

Your app can provide a rationale for users to help them understand why your app accesses location, camera, or microphone information. This rationale can appear on the new Privacy Dashboard screen, your app's permissions screen, or both.

Bluetooth permissions

Android 12 introduces the `BLUETOOTH_SCAN`, `BLUETOOTH_ADVERTISE`, and `BLUETOOTH_CONNECT` permissions. These permissions make it easier for apps that target Android 12 to interact with Bluetooth devices, especially for apps that don't require access to device location.

Update your app's Bluetooth permission declarations

To prepare your device for targeting Android 12 or higher, update your app's logic. Instead of declaring a legacy set of Bluetooth permissions, declare a more modern set of Bluetooth permissions.

Permission group lookup

On Android 12 or higher, you can query how the system organizes platform-provided permissions into permission groups:

- To determine the permission group into which the system has placed a platform-defined permission, call `getGroupOfPlatformPermission()`.
- To determine the platform-defined permissions that the system has placed into a particular permission group, call `getPlatformPermissionsForGroup()`.

Hide application overlay windows

To give developers more control over what users see when they interact with the developer's app, Android 12 introduces the ability to hide overlay windows that are drawn by apps that have the `SYSTEM_ALERT_WINDOW` permission.

After declaring the `HIDE_OVERLAY_WINDOWS` permission, an app can call `setHideOverlayWindows()` to indicate that all windows of type `TYPE_APPLICATION_OVERLAY` should be hidden when the app's own window is visible. Apps might choose to do this when displaying sensitive screens, such as transaction confirmation flows.

Apps that show windows of type `TYPE_APPLICATION_OVERLAY` should consider alternatives that may be more appropriate for their use case, such as picture-in-picture or bubbles.

Known signers permission protection flag

Starting in Android 12, the `knownCerts` attribute for signature-level permissions allows you to refer to the digests of known signing certificates at declaration time.

Your app can declare this attribute and use the `knownSigner` flag to allow devices and apps to grant signature permissions to other apps, without having to sign the apps at the time of device manufacturing and shipment.

Device properties attestation

Android 12 expands the set of apps that can verify the device properties that are in an attestation certificate when these apps generate a new key.

As of Android 9 (API level 28), device policy owners (DPOs) that use Keymaster 4.0 or higher can verify the device properties in these attestation certificates. Starting in Android 12, any app that targets Android 12 (API level 31) or higher can perform this verification using the `setDevicePropertiesAttestationIncluded()` method.

The generated device properties include the following `Build` fields:

- `BRAND`
- `DEVICE`
- `MANUFACTURER`
- `MODEL`
- `PRODUCT`

Secure lockscreen notification actions

Starting in Android 12, the `Notification.Action.Builder` class supports the `setAuthenticationRequired()` method, which allows your app to require that a device is unlocked before your app invokes a given notification

action. This method helps add an extra layer of security to notifications on locked devices.

Localizable strings for BiometricPrompt

Android 12 introduces new APIs to help you improve your app's biometric authentication user experience. The new `BiometricManager.Strings` nested class includes the `getButtonLabel()`, `getPromptMessage()`, and `getSettingName()` methods, which let your app retrieve a user-readable and localized button label, prompt message, or app setting name. Use these labels to create more precise user-facing instructions that are specific to the biometric authentication methods used, such as “Use face unlock” or “Use your fingerprint to continue”.

Compatible media transcoding

Starting in Android 12 (API level 31), the system can automatically transcode HEVC(H.265) and HDR (HDR10 and HDR10+) videos recorded on the device to AVC (H.264), a format which is widely compatible with standard players. This takes advantage of modern codecs when they are available without sacrificing compatibility with older applications.

See compatible media transcoding for more details.

Performance class

Android 12 introduces a standard called *performance class*. A performance class specifies hardware capabilities beyond Android's baseline requirements. Each Android device declares the performance class that it supports. Developers can check the device's performance class at runtime and provide upgraded experiences that take full advantage of the device’s capabilities.

See Performance class for more details.

Video encoding improvements

Android 12 defines a standard set of keys for controlling the quantization parameter (QP) value for video encoding, allowing developers to avoid vendor-specific code.

The new keys are available in the `MediaFormat` API and also in the NDK Media library.

Starting with Android 12 video encoders enforce a minimum quality threshold. This guarantees that users don't experience extremely low quality when encoding videos with high scene complexity.

Audio focus

Starting with Android 12 (API level 31), when an app requests audio focus while another app has the focus and is playing, the system fades out the playing app.

See Audio focus in Android 12 and higher for more details.

MediaDrm updates

In order to determine whether a secure decoder component is required with the current `MediaDrm` APIs, you must follow these steps:

1. Create a `MediaDrm`.
2. Open a session to obtain a session id.
3. Create a `MediaCrypto` using the session id.
4. Call `MediaCrypto.requiresSecureDecoderComponent(mimeType)`.

With the new methods `requiresSecureDecoder(@NonNull String mime)` and `requiresSecureDecoder(@NonNull String mime, @SecurityLevel int level)` you can determine this as soon as you create a `MediaDrm`.

Camera

Camera2 vendor extensions

Many of our device manufacturer partners have built custom camera extensions—such as Bokeh, HDR, Night mode, and others—that they want apps to use to create differentiated experiences on their devices. The CameraX library already supports these custom vendor extensions. In Android 12, these vendor extensions are now exposed directly in the platform.

This addition helps apps that have complex `Camera2` implementations take advantage of vendor extensions without having to make significant changes to legacy code. The Camera2 Extension APIs expose exactly the same set of extensions as in CameraX, and those are already supported on many different devices, so you can use them without any additional configuration.

For more information, see `CameraExtensionCharacteristics`.

Quad bayer camera sensor support

Many Android devices today ship with ultra high-resolution camera sensors, typically with Quad or Nona Bayer patterns, and these offer great flexibility in terms of image quality and low-light performance. Android 12 introduces new platform APIs that let third-party apps take full advantage of these versatile sensors. The new APIs support the unique behavior of these sensors and take into account that they might support different stream configurations and combinations when operating in full resolution or ‘maximum resolution’ mode vs ‘default’ mode.

Graphics and images

Provide apps direct access to tombstone traces

Starting in Android 12, you can access your app's native crash tombstone as a protocol buffer through the `ApplicationExitInfo.getTraceInputStream()` method. The protocol buffer is serialized using this schema. Previously, the only way to get access to this information was through the Android Debug Bridge (adb).

For more information, see Provide apps direct access to tombstone traces

AVIF image support

Android 12 introduces support for images that use the AV1 Image File Format (AVIF). AVIF is a container format for images and sequences of images encoded using AV1. AVIF takes advantage of the intra-frame encoded content from video compression. This dramatically improves image quality for the same file size when compared to older image formats, such as JPEG. For an in-depth look at the advantages of this format, see Jake Archibald's blog post.

Easier blurs, color filters, and other effects

Android 12 adds the new `RenderEffect` that applies common graphics effects such as blurs, color filters, Android shader effects, and more to `View` s and rendering hierarchies. Effects can be combined as either chain effects (which compose an inner and outer effect) or blended effects. Different Android devices may or may not support the feature due to limited processing power.

Effects can also be applied to the underlying `RenderNode` for `View` s by calling `View.setRenderEffect(RenderEffect)` .

To implement a `RenderEffect` :

```
view.setRenderEffect(RenderEffect.createBlurEffect(radiusX, radiusY, SHADER_TILE_MODE))
```

Native animated image decoding

In Android 12, the NDK `ImageDecoder` API has been expanded to decode all frames and timing data from images that use the animated GIF and animated WebP file formats. When it was introduced in Android 11, this API decoded only the first image from animations in these formats.

Use `ImageDecoder` instead of third-party libraries to further decrease APK size and benefit from future updates related to security and performance.

For more details on the API, refer to the API reference and the sample on GitHub.

Connectivity

Keeping companion apps awake

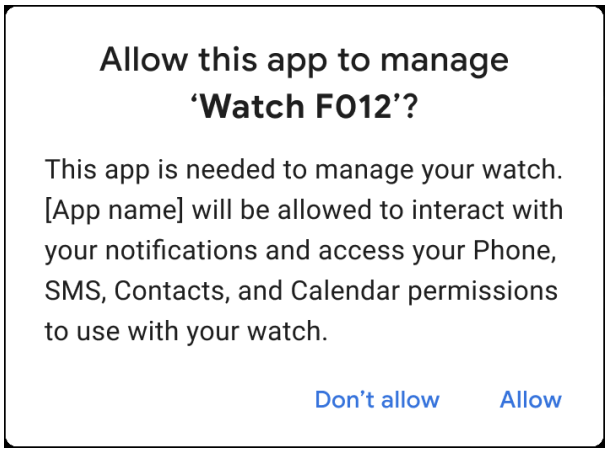
To support the need of companion apps to stay running to manage the device, Android 12 introduces APIs that do the following:

- Enable you to wake an app when a companion device is within range.
- Guarantee that the process will continue running while the device stays within range.

To use the APIs, your devices must be connected using Companion Device Manager. For more information, see `CompanionDeviceManager.startObservingDevicePresence()` and

`CompanionDeviceService.onDeviceAppeared()`.

Companion Device Manager profiles



A permissions dialog that uses a companion device profile to request multiple permissions in a single request.

Partner apps on Android 12 (API level 31) and higher can use companion device profiles when connecting to a watch. Using a profile simplifies the enrollment process by bundling the granting of a device-type-specific set of permissions into one step.

The bundled permissions are granted to the companion app once the device connects, and last only while the device is associated. Deleting the app or removing the association removes the permissions.

For more information, see `AssociationRequest.Builder.setDeviceProfile()`.

Bandwidth estimation improvements

In Android 12, the bandwidth estimation capabilities provided by `getLinkDownstreamBandwidthKbps()` and `getLinkUpstreamBandwidthKbps()` are improved for both Wi-Fi and cellular connectivity. The values returned now represent the user’s all-time weighted average throughput per carrier or WiFi SSID, network type, and signal level, across all applications on the device. This can return a more-accurate and realistic estimate of expected throughput, provide estimates on a cold start of your application, and requires fewer cycles when compared to using other throughput estimation methods.

Wi-Fi Aware (NAN) enhancements

Android 12 adds some enhancements to Wi-Fi Aware:

- On devices running Android 12 (API level 31) and higher, you can use the `onServiceLost()` callback to be alerted when your app has lost a discovered service due to the service stopping or moving out of range.
- The way that multiple data-paths (NAN Data Paths) are set up is changing to be more efficient. Earlier versions used L2 messaging to exchange peer information of the initiators, which introduced latency. On devices running Android 12 and higher, the responder (server) can be configured to accept any peer—that

is, it doesn't need to know the initiator information upfront. This speeds up datapath bringup and enables multiple point-to-point links with only one network request.
- To prevent the framework from rejecting discovery or connection requests due to running out of resources, on devices running Android 12 and higher, you can call `WifiAwareManager.getAvailableAwareResources()`. This method's return value lets you get the number of available data paths, the number of available publish sessions, and the number of available subscribe sessions.

Concurrent Peer-to-Peer + Internet Connection

When devices targeting Android 12 (API level 31) and higher run on devices with hardware support, using Peer-to-peer connections will not disconnect your existing Wi-Fi connection when creating the connection to the peer device. To check for support for this feature, use `WifiManager.isMultiStaConcurrencySupported()`.

Enable screen off for NFC payments

In apps that target Android 12 and higher, you can enable NFC payments without the device's screen on by setting `requireDeviceScreenOn` to `false`. For more information about NFC payments with screen off or locked, see Screen off and lock-screen behavior.

Storage

Android 12 introduces the following storage management capabilities:

- Media store support for `MediaDocumentsProvider` when your app retrieves a media URI that is equivalent to a given documents provider URI.
- A directory for voice recordings.
- The `MANAGE_MEDIA` permission, which allows an app to perform media management operations without showing a confirmation dialog to the user for each operation.
- Apps that have both the `MANAGE_EXTERNAL_STORAGE` permission and the `QUERY_ALL_PACKAGES` permission—such as file management apps—can invoke a custom activity for managing another app's storage space, provided that the other app creates the custom activity.

Core functionality

Automatic app updates

Android 12 introduces the `setRequireUserAction()` method for apps that use the `PackageInstaller` API. This method allows installer apps to perform app updates without requiring the user to confirm the action.

Device chipset information

Android 12 adds two constants to `android.os.Build` that expose the SoC chipset vendor and model information via the SDK. You can retrieve this information by calling `Build.SOC_MANUFACTURER` and `Build.SOC_MODEL` respectively.

Updates to core Java APIs

Based on requests and collaboration with developers, we've added the following core libraries in Android 12:
