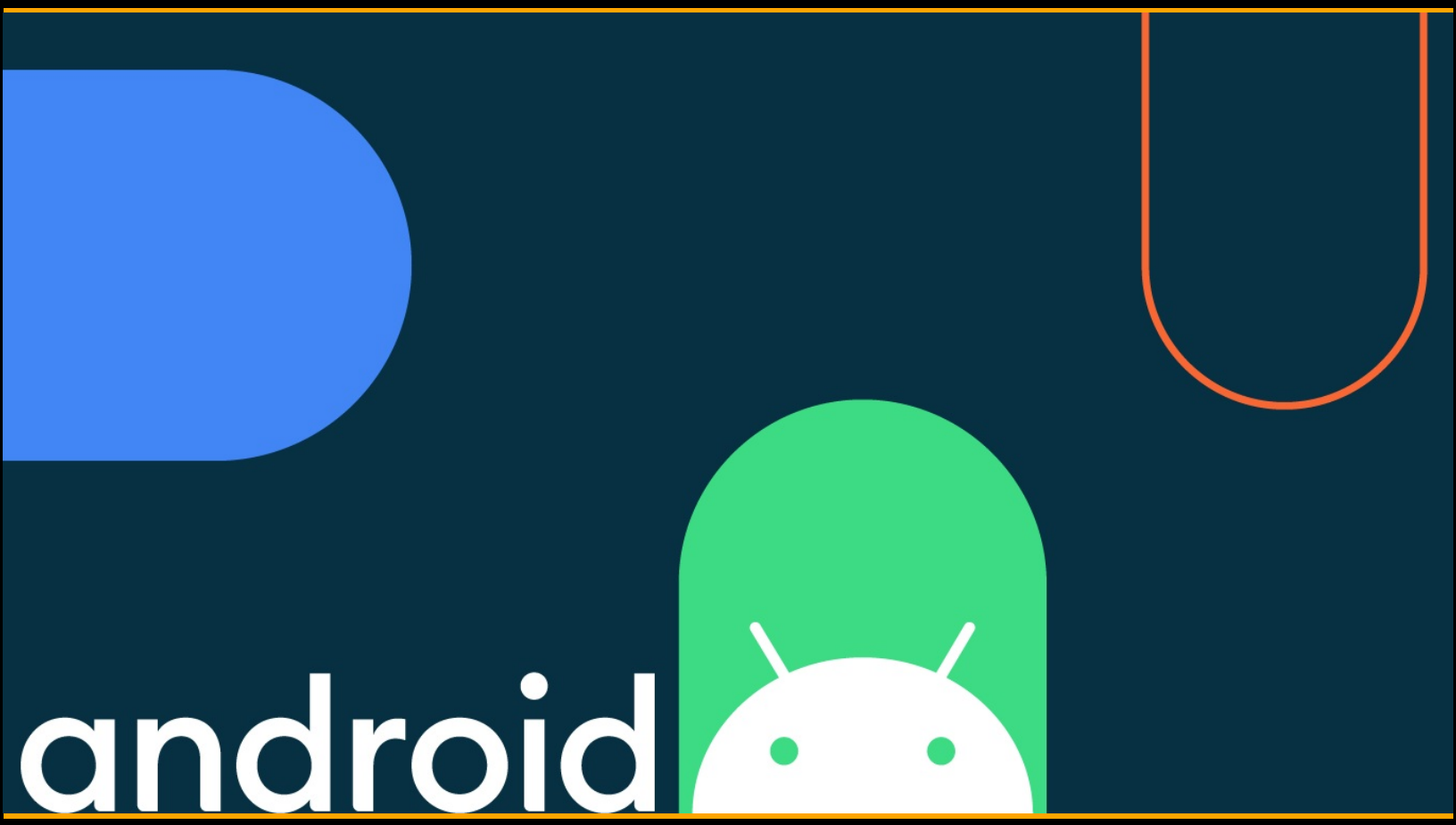
Compatibility Definition



Android 11

Last updated: September 8, 2020

Table of Contents

1. Introduction

This document enumerates the requirements that must be met in order for devices to be compatible with Android 11.

The use of “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” is per the IETF standard defined in RFC2119 .

As used in this document, a “device implementer” or “implementer” is a person or organization developing a hardware/software solution running Android 11. A “device implementation” or “implementation" is the hardware/software solution so developed.

To be considered compatible with Android 11, device implementations MUST meet the requirements presented in this Compatibility Definition, including any documents incorporated via reference.

Where this definition or the software tests described in section 10 is silent, ambiguous, or incomplete, it is the responsibility of the device implementer to ensure compatibility with existing implementations.

For this reason, the Android Open Source Project is both the reference and preferred implementation of Android. Device implementers are STRONGLY RECOMMENDED to base their implementations to the greatest extent possible on the “upstream” source code available from the Android Open Source Project. While some components can hypothetically be replaced with alternate implementations, it is STRONGLY RECOMMENDED to not follow this practice, as passing the software tests will become substantially more difficult. It is the implementer’s responsibility to ensure full behavioral compatibility with the standard Android implementation, including and beyond the Compatibility Test Suite. Finally, note that certain component substitutions and modifications are explicitly forbidden by this document.

Many of the resources linked to in this document are derived directly or indirectly from the Android SDK and will be functionally identical to the information in that SDK’s documentation. In any cases where this Compatibility Definition or the Compatibility Test Suite disagrees with the SDK documentation, the SDK documentation is considered authoritative. Any technical details provided in the linked resources throughout this document are considered by inclusion to be part of this Compatibility Definition.

1.1 Document Structure

1.1.1. Requirements by Device Type

Section 2 contains all of the requirements that apply to a specific device type. Each subsection of Section 2 is dedicated to a specific device type.

All the other requirements, that universally apply to any Android device implementations, are listed in the sections after Section 2 . These requirements are referenced as "Core Requirements" in this document.

1.1.2. Requirement ID

Requirement ID is assigned for MUST requirements.

- The ID is assigned for MUST requirements only.
- STRONGLY RECOMMENDED requirements are marked as [SR] but ID is not assigned.
- The ID consists of : Device Type ID - Condition ID - Requirement ID (e.g. C-0-1).

Each ID is defined as below:

- Device Type ID (see more in 2. Device Types)
  - C: Core (Requirements that are applied to any Android device implementations)
  - H: Android Handheld device
  - T: Android Television device
  - A: Android Automotive implementation
  - W: Android Watch implementation
  - Tab: Android Tablet implementation
- Condition ID
  - When the requirement is unconditional, this ID is set as 0.
  - When the requirement is conditional, 1 is assigned for the 1st condition and the number increments by 1 within the same section and the same device type.

- Requirement ID
  - This ID starts from 1 and increments by 1 within the same section and the same condition.

1.1.3. Requirement ID in Section 2

The Requirement ID in Section 2 starts with the corresponding section ID that is followed by the Requirement ID described above.

- The ID in Section 2 consists of : Section ID / Device Type ID - Condition ID - Requirement ID (e.g. 7.4.3/A-0-1).

2. Device Types

While the Android Open Source Project provides a software stack that can be used for a variety of device types and form factors, there are a few device types that have a relatively better established application distribution ecosystem.

This section describes those device types, and additional requirements and recommendations applicable for each device type.

All Android device implementations that do not fit into any of the described device types MUST still meet all requirements in the other sections of this Compatibility Definition.

2.1 Device Configurations

For the major differences in hardware configuration by device type, see the device-specific requirements that follow in this section.

2.2. Handheld Requirements

An **Android Handheld device** refers to an Android device implementation that is typically used by holding it in the hand, such as an mp3 player, phone, or tablet.

Android device implementations are classified as a Handheld if they meet all the following criteria:

- Have a power source that provides mobility, such as a battery.
- Have a physical diagonal screen size in the range of 3.3 inches (or 2.5 inches for devices which launched on an API level earlier than Android 11) to 8 inches.

The additional requirements in the rest of this section are specific to Android Handheld device implementations.

Note: Requirements that do not apply to Android Tablet devices are marked with an *.

2.2.1. Hardware

Handheld device implementations:

- [7.1 .1.1/H-0-1] MUST have at least one Android-compatible display that meets all requirements described on this document.
- [7.1 .1.3/H-SR] Are STRONGLY RECOMMENDED to provide users an affordance to change the display size (screen density).

If Handheld device implementations support software screen rotation, they:

- [7.1 .1.1/H-1-1]* MUST make the logical screen that is made available for third party applications be at least 2 inches on the short edge(s) and 2.7 inches on the long edge(s). Devices which launched on an API level earlier than that of this document are exempted from this requirement.

If Handheld device implementations do not support software screen rotation, they:

- [7.1 .1.1/H-2-1]* MUST make the logical screen that is made available for third party applications be at least 2.7 inches on the short edge(s). Devices which launched on an API level earlier than that of this document are exempted from this requirement.

If Handheld device implementations claim support for high dynamic range displays through

Configuration.isScreenHdr() , they:

- [7.1 .4.5/H-1-1] MUST advertise support for the EGL_EXT_gl_colorspace_bt2020_pq , EGL_EXT_surface_SMPTE2086_metadata , EGL_EXT_surface_CTA861_3_metadata , VK_EXT_swapchain_colorspace , and VK_EXT_hdr_metadata extensions.

Handheld device implementations:

- [7.1 .4.6/H-0-1] MUST report whether the device supports the GPU profiling capability via a system property graphics.gpu.profiler.support .

If Handheld device implementations declare support via a system property graphics.gpu.profiler.support , they:

- [7.1 .4.6/H-1-1] MUST report as output a protobuf trace that complies with the schema for GPU counters and GPU renderstages defined in the Perfetto documentation .
- [7.1 .4.6/H-1-2] MUST report conformant values for the device's GPU counters following the gpu counter trace packet proto .
- [7.1 .4.6/H-1-3] MUST report conformant values for the device's GPU RenderStages following the render stage trace packet proto .
- [7.1 .4.6/H-1-4] MUST report a GPU Frequency tracepoint as specified by the format: power/gpu_frequency .

Handheld device implementations:

- [7.1 .5/H-0-1] MUST include support for legacy application compatibility mode as implemented by the upstream Android open source code. That is, device implementations MUST NOT alter the triggers or thresholds at which compatibility mode is activated, and MUST NOT alter the behavior of the compatibility mode itself.
- [7.2 .1/H-0-1] MUST include support for third-party Input Method Editor (IME) applications.
- [7.2 .3/H-0-3] MUST provide the Home function on all the Android-compatible displays that provide the home screen.
- [7.2 .3/H-0-4] MUST provide the Back function on all the Android-compatible displays and the Recents function on at least one of the Android-compatible displays.
- [7.2 .3/H-0-2] MUST send both the normal and long press event of the Back function (KEYCODE_BACK) to the foreground application. These events MUST NOT be consumed by the system and CAN be triggered by outside of the Android device (e.g. external hardware keyboard connected to the Android device).
- [7.2 .4/H-0-1] MUST support touchscreen input.
- [7.2 .4/H-SR] Are STRONGLY RECOMMENDED to launch the user-selected assist app, in other words the app that implements VoiceInteractionService, or an activity handling the ACTION_ASSIST on long-press of KEYCODE_MEDIA_PLAY_PAUSE or KEYCODE_HEADSETHOOK if the foreground activity does not handle those long-press events.
- [7.3 .1/H-SR] Are STRONGLY RECOMMENDED to include a 3-axis accelerometer.

If Handheld device implementations include a 3-axis accelerometer, they:

- [7.3 .1/H-1-1] MUST be able to report events up to a frequency of at least 100 Hz.

If Handheld device implementations include a GPS/GNSS receiver and report the capability to applications through the android.hardware.location.gps feature flag, they:

- [7.3 .3/H-2-1] MUST report GNSS measurements, as soon as they are found, even if a location calculated from GPS/GNSS is not yet reported.
- [7.3 .3/H-2-2] MUST report GNSS pseudoranges and pseudorange rates, that, in open-sky conditions after determining the location, while stationary or moving with less than 0.2 meter per second squared of acceleration, are sufficient to calculate position within 20 meters, and speed within 0.2 meters per second, at least 95% of the time.

If Handheld device implementations include a 3-axis gyroscope, they:

- [7.3 .4/H-3-1] MUST be able to report events up to a frequency of at least 100 Hz.
- [7.3 .4/H-3-2] MUST be capable of measuring orientation changes up to 1000 degrees per second.

Handheld device implementations that can make a voice call and indicate any value other than PHONE_TYPE_NONE in getPhoneType :

- [7.3 .8/H] SHOULD include a proximity sensor.

Handheld device implementations:

- [7.3 .11/H-SR] Are RECOMMENDED to support pose sensor with 6 degrees of freedom.
- [7.4 .3/H] SHOULD include support for Bluetooth and Bluetooth LE.

If Handheld device implementations include a metered connection, they:

- [7.4 .7/H-1-1] MUST provide the data saver mode.

If Handheld device implementations include a logical camera device that lists capabilities using CameraMetadata.REQUEST_AVAILABLE_CAPABILITIES_LOGICAL_MULTI_CAMERA , they:

- [7.5 .4/H-1-1] MUST have normal field of view (FOV) by default and it MUST be between 50 and 90 degrees.

Handheld device implementations:

- [7.6 .1/H-0-1] MUST have at least 4 GB of non-volatile storage available for application private data (a.k.a. "/data" partition).
- [7.6 .1/H-0-2] MUST return “true” for ActivityManager.isLowRamDevice() when there is less than 1GB of memory available to the kernel and userspace.

If Handheld device implementations declare support of only a 32-bit ABI:

- [7.6 .1/H-1-1] The memory available to the kernel and userspace MUST be at least 416MB if the default display uses framebuffer resolutions up to qHD (e.g. FWVGA).
- [7.6 .1/H-2-1] The memory available to the kernel and userspace MUST be at least 592MB if the default display uses framebuffer resolutions up to HD+ (e.g. HD, WSVGA).
- [7.6 .1/H-3-1] The memory available to the kernel and userspace MUST be at least 896MB if the default display uses framebuffer resolutions up to FHD (e.g. WSXGA+).
- [7.6 .1/H-4-1] The memory available to the kernel and userspace MUST be at least 1344MB if the default display uses framebuffer resolutions up to QHD (e.g. QWXGA).

If Handheld device implementations declare support of 32-bit and 64-bit ABIs:

- [7.6 .1/H-5-1] The memory available to the kernel and userspace MUST be at least 816MB if the default display uses framebuffer resolutions up to qHD (e.g. FWVGA).
- [7.6 .1/H-6-1] The memory available to the kernel and userspace MUST be at least 944MB if the default display uses framebuffer resolutions up to HD+ (e.g. HD, WSVGA).
- [7.6 .1/H-7-1] The memory available to the kernel and userspace MUST be at least 1280MB if the default display uses framebuffer resolutions up to FHD (e.g. WSXGA+).
- [7.6 .1/H-8-1] The memory available to the kernel and userspace MUST be at least 1824MB if the default display uses framebuffer resolutions up to QHD (e.g. QWXGA).

Note that the "memory available to the kernel and userspace" above refers to the memory space provided in addition to any memory already dedicated to hardware components such as radio, video, and so on that are not under the kernel’s control on device implementations.

If Handheld device implementations include less than or equal to 1GB of memory available to the kernel and userspace, they:

- [7.6 .1/H-9-1] MUST declare the feature flag android.hardware.ram.low .
- [7.6 .1/H-9-2] MUST have at least 1.1 GB of non-volatile storage for application private data (a.k.a. "/data" partition).

If Handheld device implementations include more than 1GB of memory available to the kernel and userspace, they:

- [7.6 .1/H-10-1] MUST have at least 4GB of non-volatile storage available for application private data (a.k.a. "/data" partition).

- SHOULD declare the feature flag android.hardware.ram.normal .

Handheld device implementations:

- [7.6 .2/H-0-1] MUST NOT provide an application shared storage smaller than 1 GiB.
- [7.7 .1/H] SHOULD include a USB port supporting peripheral mode.

If handheld device implementations include a USB port supporting peripheral mode, they:

- [7.7 .1/H-1-1] MUST implement the Android Open Accessory (AOA) API.

If Handheld device implementations include a USB port supporting host mode, they:

- [7.7 .2/H-1-1] MUST implement the USB audio class as documented in the Android SDK documentation.

Handheld device implementations:

- [7.8 .1/H-0-1] MUST include a microphone.
- [7.8 .2/H-0-1] MUST have an audio output and declare android.hardware.audio.output .

If Handheld device implementations are capable of meeting all the performance requirements for supporting VR mode and include support for it, they:

- [7.9 .1/H-1-1] MUST declare the android.hardware.vr.high_performance feature flag.
- [7.9 .1/H-1-2] MUST include an application implementing android.service.vr.VrListenerService that can be enabled by VR applications via android.app.Activity#setVrModeEnabled .

If Handheld device implementations include one or more USB-C port(s) in host mode and implement (USB audio class), in addition to requirements in section 7.7.2 , they:

- [7.8 .2.2/H-1-1] MUST provide the following software mapping of HID codes:

Function	Mappings	Context	Behavior
A	**HID usage page** : 0x0C **HID usage** : 0x0CD **Kernel key** : KEY_PLAYPAUSE **Android key** : KEYCODE_MEDIA_PLAY_PAUSE	Media playback	**Input** : Short press **Output** : Play or pause
			Input : Long press **Output** : Launch voice command **Sends** : android.speech.action.VOICE_SEARCH_HANDS_FREE if the device is locked or its screen is off. Sends android.speech.RecognizerIntent.ACTION_WEB_SEARCH otherwise
		Incoming call	**Input** : Short press **Output** : Accept call
			Input : Long press **Output** : Reject call
		Ongoing call	**Input** : Short press **Output** : End call
			Input : Long press **Output** : Mute or unmute microphone
B	**HID usage page** : 0x0C **HID usage** : 0x0E9 **Kernel key** : KEY_VOLUMEUP **Android key** : VOLUME_UP	Media playback, Ongoing call	**Input** : Short or long press **Output** : Increases the system or headset volume
C	**HID usage page** : 0x0C **HID usage** : 0x0EA **Kernel key** : KEY_VOLUMEDOWN **Android key** : VOLUME_DOWN	Media playback, Ongoing call	**Input** : Short or long press **Output** : Decreases the system or headset volume
	HID usage page : 0x0C		

D	**HID usage** : 0x0CF **Kernel key** : KEY_VOICECOMMAND **Android key** : KEYCODE_VOICE_ASSIST	All. Can be triggered in any instance.	**Input** : Short or long press **Output** : Launch voice command

- [7.8 .2.2/H-1-2] MUST trigger ACTION_HEADSET_PLUG upon a plug insert, but only after the USB audio interfaces and endpoints have been properly enumerated in order to identify the type of terminal connected.

When the USB audio terminal types 0x0302 is detected, they:

- [7.8 .2.2/H-2-1] MUST broadcast Intent ACTION_HEADSET_PLUG with "microphone" extra set to 0.

When the USB audio terminal types 0x0402 is detected, they:

- [7.8 .2.2/H-3-1] MUST broadcast Intent ACTION_HEADSET_PLUG with "microphone" extra set to 1.

When API AudioManager.getDevices() is called while the USB peripheral is connected they:

- [7.8 .2.2/H-4-1] MUST list a device of type AudioDeviceInfo.TYPE_USB_HEADSET and role isSink() if the USB audio terminal type field is 0x0302.
- [7.8 .2.2/H-4-2] MUST list a device of type AudioDeviceInfo.TYPE_USB_HEADSET and role isSink() if the USB audio terminal type field is 0x0402.
- [7.8 .2.2/H-4-3] MUST list a device of type AudioDeviceInfo.TYPE_USB_HEADSET and role isSource() if the USB audio terminal type field is 0x0402.
- [7.8 .2.2/H-4-4] MUST list a device of type AudioDeviceInfo.TYPE_USB_DEVICE and role isSink() if the USB audio terminal type field is 0x603.
- [7.8 .2.2/H-4-5] MUST list a device of type AudioDeviceInfo.TYPE_USB_DEVICE and role isSource() if the USB audio terminal type field is 0x604.
- [7.8 .2.2/H-4-6] MUST list a device of type AudioDeviceInfo.TYPE_USB_DEVICE and role isSink() if the USB audio terminal type field is 0x400.
- [7.8 .2.2/H-4-7] MUST list a device of type AudioDeviceInfo.TYPE_USB_DEVICE and role isSource() if the USB audio terminal type field is 0x400.
- [7.8 .2.2/H-SR] Are STRONGLY RECOMMENDED upon connection of a USB-C audio peripheral, to perform enumeration of USB descriptors, identify terminal types and broadcast Intent ACTION_HEADSET_PLUG in less than 1000 milliseconds.

If Handheld device implementations include at least one haptic actuator, they:

- [7.10 /H-SR]* Are STRONGLY RECOMMENDED NOT to use an eccentric rotating mass (ERM) haptic actuator(vibrator).
- [7.10 /H]* SHOULD position the placement of the actuator near the location where the device is typically held or touched by hands.
- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to implement all public constants for clear haptics in android.view.HapticFeedbackConstants namely (CLOCK_TICK, CONTEXT_CLICK, KEYBOARD_PRESS, KEYBOARD_RELEASE, KEYBOARD_TAP, LONG_PRESS, TEXT_HANDLE_MOVE, VIRTUAL_KEY, VIRTUAL_KEY_RELEASE, CONFIRM, REJECT, GESTURE_START and GESTURE_END).
- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to implement all public constants for clear haptics in android.os.VibrationEffect namely (EFFECT_TICK, EFFECT_CLICK, EFFECT_HEAVY_CLICK and EFFECT_DOUBLE_CLICK) and all public constants for rich haptics in android.os.VibrationEffect.Composition namely (PRIMITIVE_CLICK and PRIMITIVE_TICK).
- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to use these linked haptic constants mappings .
- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to follow quality assessment for createOneShot() and createWaveform() API's.
- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to verify the capabilities for amplitude scalability by running android.os.Vibrator.hasAmplitudeControl() .

Linear resonant actuator (LRA) is a single mass spring system which has a dominant resonant

frequency where the mass translates in the direction of desired motion.

If Handheld device implementations include at least one linear resonant actuator, they:

- [7.10 /H]* SHOULD move the haptic actuator in the X-axis of portrait orientation.

If Handheld device implementations have a haptic actuator which is X-axis Linear resonant actuator (LRA), they:

- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to have the resonant frequency of the X-axis LRA be under 200 Hz.

If handheld device implementations follow haptic constants mapping, they:

- [7.10 /H-SR]* Are STRONGLY RECOMMENDED to perform a quality assessment for haptic constants.

2.2.2. Multimedia

Handheld device implementations MUST support the following audio encoding and decoding formats and make them available to third-party applications:

- [5.1 /H-0-1] AMR-NB
- [5.1 /H-0-2] AMR-WB
- [5.1 /H-0-3] MPEG-4 AAC Profile (AAC LC)
- [5.1 /H-0-4] MPEG-4 HE AAC Profile (AAC+)
- [5.1 /H-0-5] AAC ELD (enhanced low delay AAC)

Handheld device implementations MUST support the following video encoding formats and make them available to third-party applications:

- [5.2 /H-0-1] H.264 AVC
- [5.2 /H-0-2] VP8

Handheld device implementations MUST support the following video decoding formats and make them available to third-party applications:

- [5.3 /H-0-1] H.264 AVC
- [5.3 /H-0-2] H.265 HEVC
- [5.3 /H-0-3] MPEG-4 SP
- [5.3 /H-0-4] VP8
- [5.3 /H-0-5] VP9

2.2.3. Software

Handheld device implementations:

- [3.2.3.1 /H-0-1] MUST have an application that handles the ACTION_GET_CONTENT , ACTION_OPEN_DOCUMENT , ACTION_OPEN_DOCUMENT_TREE , and ACTION_CREATE_DOCUMENT intents as described in the SDK documents, and provide the user affordance to access the document provider data by using DocumentsProvider API.
- [3.2.3.1 /H-0-2]* MUST preload one or more applications or service components with an intent handler, for all the public intent filter patterns defined by the following application intents listed here .
- [3.2.3.1 /H-SR] Are STRONGLY RECOMMENDED to preload an email application which can handle ACTION_SENDTO or ACTION_SEND or ACTION_SEND_MULTIPLE intents to send an email.
- [3.4 .1/H-0-1] MUST provide a complete implementation of the android.webkit.Webview API.
- [3.4 .2/H-0-1] MUST include a standalone Browser application for general user web browsing.
- [3.8 .1/H-SR] Are STRONGLY RECOMMENDED to implement a default launcher that supports in-app pinning of shortcuts, widgets and widgetFeatures .
- [3.8 .1/H-SR] Are STRONGLY RECOMMENDED to implement a default launcher that provides quick access to the additional shortcuts provided by third-party apps through the

ShortcutManager API.

- [3.8 .1/H-SR] Are STRONGLY RECOMMENDED to include a default launcher app that shows badges for the app icons.
- [3.8 .2/H-SR] Are STRONGLY RECOMMENDED to support third-party app widgets.
- [3.8 .3/H-0-1] MUST allow third-party apps to notify users of notable events through the Notification and NotificationManager API classes.
- [3.8 .3/H-0-2] MUST support rich notifications.
- [3.8 .3/H-0-3] MUST support heads-up notifications.
- [3.8 .3/H-0-4] MUST include a notification shade, providing the user the ability to directly control (e.g. reply, snooze, dismiss, block) the notifications through user affordance such as action buttons or the control panel as implemented in the AOSP.
- [3.8 .3/H-0-5] MUST display the choices provided through RemoteInput.Builder setChoices() in the notification shade.
- [3.8 .3/H-SR] Are STRONGLY RECOMMENDED to display the first choice provided through RemoteInput.Builder setChoices() in the notification shade without additional user interaction.
- [3.8 .3/H-SR] Are STRONGLY RECOMMENDED to display all the choices provided through RemoteInput.Builder setChoices() in the notification shade when the user expands all notifications in the notification shade.
- [3.8 .3.1/H-SR] Are STRONGLY RECOMMENDED to display actions for which Notification.Action.Builder.setContextual is set as true in-line with the replies displayed by Notification.Remoteinput.Builder.setChoices .
- [3.8 .4/H-SR] Are STRONGLY RECOMMENDED to implement an assistant on the device to handle the Assist action .

If Handheld device implementations support Assist action, they:

- [3.8 .4/H-SR] Are STRONGLY RECOMMENDED to use long press on HOME key as the designated interaction to launch the assist app as described in section 7.2.3 . MUST launch the user-selected assist app, in other words the app that implements VoiceInteractionService , or an activity handling the ACTION_ASSIST intent.

If Handheld device implementations support conversation notifications and group them into a separate section from alerting and silent non-conversation notifications, they:

- [3.8 .4/H-1-1]* MUST display conversation notifications ahead of non conversation notifications with the exception of ongoing foreground service notifications and importance:high notifications.

If Android Handheld device implementations support a lock screen, they:

- [3.8 .10/H-1-1] MUST display the Lock screen Notifications including the Media Notification Template.

If Handheld device implementations support a secure lock screen, they:

- [3.9 /H-1-1] MUST implement the full range of device administration policies defined in the Android SDK documentation.
- [3.9 /H-1-2] MUST declare the support of managed profiles via the android.software.managed_users feature flag, except when the device is configured so that it would report itself as a low RAM device or so that it allocates internal (non-removable) storage as shared storage.

If Handheld device implementations include support for ControlsProviderService and Control APIs and allow third-party applications to publish device controls , then they:

- [3.8 .16/H-1-1] MUST declare the feature flag android.software.controls and set it to true .
- [3.8 .16/H-1-2] MUST provide a user affordance with the ability to add, edit, select, and operate the user's favorite device controls from the controls registered by the third-party applications through the ControlsProviderService and the Control APIs.
- [3.8 .16/H-1-3] MUST provide access to this user affordance within three interactions from a default Launcher.
- [3.8 .16/H-1-4] MUST accurately render in this user affordance the name and icon of each third-party app that provides controls via the ControlsProviderService API as well as any specified fields provided by the Control APIs.

Conversely, If Handheld device implementations do not implement such controls, they:

- [3.8 .16/H-2-1] MUST report null for the ControlsProviderService and the Control APIs.
- [3.8 .16/H-2-2] MUST declare the feature flag android.software.controls and set it to false .

Handheld device implementations:

- [3.10 /H-0-1] MUST support third-party accessibility services.
- [3.10 /H-SR] Are STRONGLY RECOMMENDED to preload accessibility services on the device comparable with or exceeding functionality of the Switch Access and TalkBack (for languages supported by the preinstalled Text-to-speech engine) accessibility services as provided in the talkback open source project .
- [3.11 /H-0-1] MUST support installation of third-party TTS engines.
- [3.11 /H-SR] Are STRONGLY RECOMMENDED to include a TTS engine supporting the languages available on the device.
- [3.13 /H-SR] Are STRONGLY RECOMMENDED to include a Quick Settings UI component.

If Android handheld device implementations declare FEATURE_BLUETOOTH or FEATURE_WIFI support, they:

- [3.16 /H-1-1] MUST support the companion device pairing feature.

If the navigation function is provided as an on-screen, gesture-based action:

- [7.2 .3/H] The gesture recognition zone for the Home function SHOULD be no higher than 32 dp in height from the bottom of the screen.

If Handheld device implementations provide a navigation function as a gesture from anywhere on the left and right edges of the screen:

- [7.2 .3/H-0-1] The navigation function's gesture area MUST be less than 40 dp in width on each side. The gesture area SHOULD be 24 dp in width by default.

2.2.4. Performance and Power

- [8.1 /H-0-1] **Consistent frame latency** . Inconsistent frame latency or a delay to render frames MUST NOT happen more often than 5 frames in a second, and SHOULD be below 1 frames in a second.
- [8.1 /H-0-2] **User interface latency** . Device implementations MUST ensure low latency user experience by scrolling a list of 10K list entries as defined by the Android Compatibility Test Suite (CTS) in less than 36 secs.
- [8.1 /H-0-3] **Task switching** . When multiple applications have been launched, re-launching an already-running application after it has been launched MUST take less than 1 second.

Handheld device implementations:

- [8.2 /H-0-1] MUST ensure a sequential write performance of at least 5 MB/s.
- [8.2 /H-0-2] MUST ensure a random write performance of at least 0.5 MB/s.
- [8.2 /H-0-3] MUST ensure a sequential read performance of at least 15 MB/s.
- [8.2 /H-0-4] MUST ensure a random read performance of at least 3.5 MB/s.

If Handheld device implementations include features to improve device power management that are included in AOSP or extend the features that are included in AOSP, they:

- [8.3 /H-1-1] MUST provide user affordance to enable and disable the battery saver feature.
- [8.3 /H-1-2] MUST provide user affordance to display all apps that are exempted from App Standby and Doze power-saving modes.

Handheld device implementations:

- [8.4 /H-0-1] MUST provide a per-component power profile that defines the current consumption value for each hardware component and the approximate battery drain caused by the components over time as documented in the Android Open Source Project site.

- [8.4 /H-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [8.4 /H-0-3] MUST report CPU power consumption per each process's UID. The Android Open Source Project meets the requirement through the uid_cputime kernel module implementation.
- [8.4 /H-0-4] MUST make this power usage available via the adb shell dumpsys batterystats shell command to the app developer.
- [8.4 /H] SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.

If Handheld device implementations include a screen or video output, they:

- [8.4 /H-1-1] MUST honor the android.intent.action.POWER_USAGE_SUMMARY intent and display a settings menu that shows this power usage.

2.2.5. Security Model

Handheld device implementations:

- [9.1 /H-0-1] MUST allow third-party apps to access the usage statistics via the android.permission.PACKAGE_USAGE_STATS permission and provide a user-accessible mechanism to grant or revoke access to such apps in response to the android.settings.ACTION_USAGE_ACCESS_SETTINGS intent.

Handheld device implementations (* Not applicable for Tablet):

- [9.11 /H-0-2]* MUST back up the keystore implementation with an isolated execution environment.
- [9.11 /H-0-3]* MUST have implementations of RSA, AES, ECDSA, and HMAC cryptographic algorithms and MD5, SHA1, and SHA-2 family hash functions to properly support the Android Keystore system's supported algorithms in an area that is securely isolated from the code running on the kernel and above. Secure isolation MUST block all potential mechanisms by which kernel or userspace code might access the internal state of the isolated environment, including DMA. The upstream Android Open Source Project (AOSP) meets this requirement by using the Trusty implementation, but another ARM TrustZone-based solution or a third-party reviewed secure implementation of a proper hypervisor-based isolation are alternative options.
- [9.11 /H-0-4]* MUST perform the lock screen authentication in the isolated execution environment and only when successful, allow the authentication-bound keys to be used. Lock screen credentials MUST be stored in a way that allows only the isolated execution environment to perform lock screen authentication. The upstream Android Open Source Project provides the Gatekeeper Hardware Abstraction Layer (HAL) and Trusty, which can be used to satisfy this requirement.
- [9.11 /H-0-5]* MUST support key attestation where the attestation signing key is protected by secure hardware and signing is performed in secure hardware. The attestation signing keys MUST be shared across large enough number of devices to prevent the keys from being used as device identifiers. One way of meeting this requirement is to share the same attestation key unless at least 100,000 units of a given SKU are produced. If more than 100,000 units of an SKU are produced, a different key MAY be used for each 100,000 units.

Note that if a device implementation is already launched on an earlier Android version, such a device is exempted from the requirement to have a keystore backed by an isolated execution environment and support the key attestation, unless it declares the android.hardware.fingerprint feature which requires a keystore backed by an isolated execution environment.

When Handheld device implementations support a secure lock screen, they:

- [9.11 /H-1-1] MUST allow the user to choose the shortest sleep timeout, that is a transition time from the unlocked to the locked state, as 15 seconds or less.
- [9.11 /H-1-2] MUST provide user affordance to hide notifications and disable all forms of authentication except for the primary authentication described in 9.11.1 Secure Lock Screen . The AOSP meets the requirement as lockdown mode.

2.2.6. Developer Tools and Options Compatibility

Handheld device implementations (* Not applicable for Tablet):

- [6.1 /H-0-1]* MUST support the shell command cmd testharness .

Handheld device implementations (* Not applicable for Tablet):

- Perfetto
  - [6.1 /H-0-2]* MUST expose a /system/bin/perfetto binary to the shell user which cmdline complies with the perfetto documentation .
  - [6.1 /H-0-3]* The perfetto binary MUST accept as input a protobuf config that complies with the schema defined in the perfetto documentation .
  - [6.1 /H-0-4]* The perfetto binary MUST write as output a protobuf trace that complies with the schema defined in the perfetto documentation .
  - [6.1 /H-0-5]* MUST provide, through the perfetto binary, at least the data sources described in the perfetto documentation .
  - [6.1 /H-0-6]* The perfetto traced daemon MUST be enabled by default (system property persist.traced.enable).

2.3. Television Requirements

An **Android Television device** refers to an Android device implementation that is an entertainment interface for consuming digital media, movies, games, apps, and/or live TV for users sitting about ten feet away (a “lean back” or “10-foot user interface”).

Android device implementations are classified as a Television if they meet all the following criteria:

- Have provided a mechanism to remotely control the rendered user interface on the display that might sit ten feet away from the user.
- Have an embedded screen display with the diagonal length larger than 24 inches OR include a video output port, such as VGA, HDMI, DisplayPort, or a wireless port for display.

The additional requirements in the rest of this section are specific to Android Television device implementations.

2.3.1. Hardware

Television device implementations:

- [7.2 .2/T-0-1] MUST support D-pad .
- [7.2 .3/T-0-1] MUST provide the Home and Back functions.
- [7.2 .3/T-0-2] MUST send both the normal and long press event of the Back function (KEYCODE_BACK) to the foreground application.
- [7.2 .6.1/T-0-1] MUST include support for game controllers and declare the android.hardware.gamepad feature flag.
- [7.2 .7/T] SHOULD provide a remote control from which users can access non-touch navigation and core navigation keys inputs.

If Television device implementations include a 3-axis gyroscope, they:

- [7.3 .4/T-1-1] MUST be able to report events up to a frequency of at least 100 Hz.
- [7.3 .4/T-1-2] MUST be capable of measuring orientation changes up to 1000 degrees per second.

Television device implementations:

- [7.4 .3/T-0-1] MUST support Bluetooth and Bluetooth LE.
- [7.6 .1/T-0-1] MUST have at least 4 GB of non-volatile storage available for application private data (a.k.a. "/data" partition).

If Television device implementations include a USB port that supports host mode, they:

- [7.5 .3/T-1-1] MUST include support for an external camera that connects through this USB port but is not necessarily always connected.

If TV device implementations are 32-bit:

- [7.6 .1/T-1-1] The memory available to the kernel and userspace MUST be at least 896MB if any of the following densities are used:

    - 400dpi or higher on small/normal screens
    - xhdpi or higher on large screens
    - tvdpi or higher on extra large screens

If TV device implementations are 64-bit:

- [7.6 .1/T-2-1] The memory available to the kernel and userspace MUST be at least 1280MB if any of the following densities are used:
    - 400dpi or higher on small/normal screens
    - xhdpi or higher on large screens
    - tvdpi or higher on extra large screens

Note that the "memory available to the kernel and userspace" above refers to the memory space provided in addition to any memory already dedicated to hardware components such as radio, video, and so on that are not under the kernel's control on device implementations.

Television device implementations:

- [7.8 .1/T] SHOULD include a microphone.
- [7.8 .2/T-0-1] MUST have an audio output and declare android.hardware.audio.output .

2.3.2. Multimedia

Television device implementations MUST support the following audio encoding and decoding formats and make them available to third-party applications:

- [5.1 /T-0-1] MPEG-4 AAC Profile (AAC LC)
- [5.1 /T-0-2] MPEG-4 HE AAC Profile (AAC+)
- [5.1 /T-0-3] AAC ELD (enhanced low delay AAC)

Television device implementations MUST support the following video encoding formats and make them available to third-party applications:

- [5.2 /T-0-1] H.264
- [5.2 /T-0-2] VP8

Television device implementations:

- [5.2 .2/T-SR] Are STRONGLY RECOMMENDED to support H.264 encoding of 720p and 1080p resolution videos at 30 frames per second.

Television device implementations MUST support the following video decoding formats and make them available to third-party applications:

- [5.3.3 /T-0-1] MPEG-4 SP
- [5.3.4 /T-0-2] H.264 AVC
- [5.3.5 /T-0-3] H.265 HEVC
- [5.3.6 /T-0-4] VP8
- [5.3.7 /T-0-5] VP9
- [5.3.1 /T-0-6] MPEG-2

Television device implementations MUST support MPEG-2 decoding, as detailed in Section 5.3.1, at standard video frame rates and resolutions up to and including:

- [5.3.1 /T-1-1] HD 1080p at 59.94 frames per second with Main Profile High Level.
- [5.3.1 /T-1-2] HD 1080i at 59.94 frames per second with Main Profile High Level. They MUST deinterlace interlaced MPEG-2 video to its progressive equivalent (e.g. from 1080i at 59.94 frames per second to 1080p at 29.97 frames per second) and make it available to third-party applications.

Television device implementations MUST support H.264 decoding, as detailed in Section 5.3.4, at standard video frame rates and resolutions up to and including:

- [5.3.4 /T-1-1] HD 1080p at 60 frames per second with Baseline Profile
- [5.3.4 /T-1-2] HD 1080p at 60 frames per second with Main Profile

- [5.3.4 /T-1-3] HD 1080p at 60 frames per second with High Profile Level 4.2

Television device implementations with H.265 hardware decoders MUST support H.265 decoding, as detailed in Section 5.3.5, at standard video frame rates and resolutions up to and including:

- [5.3.5 /T-1-1] HD 1080p at 60 frames per second with Main Profile Level 4.1

If Television device implementations with H.265 hardware decoders support H.265 decoding and the UHD decoding profile, they:

- [5.3.5 /T-2-1] MUST support UHD 3480p at 60 frames per second with Main10 Level 5 Main Tier profile

Television device implementations MUST support VP8 decoding, as detailed in Section 5.3.6, at standard video frame rates and resolutions up to and including:

- [5.3.6 /T-1-1] HD 1080p at 60 frames per second decoding profile

Television device implementations with VP9 hardware decoders MUST support VP9 decoding, as detailed in Section 5.3.7, at standard video frame rates and resolutions up to and including:

- [5.3.7 /T-1-1] HD 1080p at 60 frames per second with profile 0 (8 bit color depth)

If Television device implementations with VP9 hardware decoders support VP9 decoding and the UHD decoding profile, they:

- [5.3.7 /T-2-1] MUST support UHD 3480p at 60 frames per second with profile 0 (8 bit color depth).
- [5.3.7 /T-2-1] Are STRONGLY RECOMMENDED to support UHD 3480p at 60 frames per second with profile 2 (10 bit color depth).

Television device implementations:

- [5.5 /T-0-1] MUST include support for system Master Volume and digital audio output volume attenuation on supported outputs, except for compressed audio passthrough output (where no audio decoding is done on the device).

If Television device implementations do not have a built in display, but instead support an external display connected via HDMI, they:

- [5.8 /T-0-1] MUST set the HDMI output mode to select the maximum resolution that can be supported with either a 50Hz or 60Hz refresh rate.
- [5.8 /T-SR] Are STRONGLY RECOMMENDED to provide a user configurable HDMI refresh rate selector.
- [5.8] SHOULD set the HDMI output mode refresh rate to either 50Hz or 60Hz, depending on the video refresh rate for the region the device is sold in.

If Television device implementations do not have a built in display, but instead support an external display connected via HDMI, they:

- [5.8 /T-1-1] MUST support HDCP 2.2.

If Television device implementations do not support UHD decoding, but instead support an external display connected via HDMI, they:

- [5.8 /T-2-1] MUST support HDCP 1.4

2.3.3. Software

Television device implementations:

- [3 /T-0-1] MUST declare the features android.software.leanback and android.hardware.type.television .
- [3.2.3.1 /T-0-1] MUST preload one or more applications or service components with an intent handler, for all the public intent filter patterns defined by the following application intents listed here .
- [3.4 .1/T-0-1] MUST provide a complete implementation of the android.webkit.Webview

API.

If Android Television device implementations support a lock screen,they:

- [3.8 .10/T-1-1] MUST display the Lock screen Notifications including the Media Notification Template.

Television device implementations:

- [3.8 .14/T-SR] Are STRONGLY RECOMMENDED to support picture-in-picture (PIP) mode multi-window.
- [3.10 /T-0-1] MUST support third-party accessibility services.
- [3.10 /T-SR] Are STRONGLY RECOMMENDED to preload accessibility services on the device comparable with or exceeding functionality of the Switch Access and TalkBack (for languages supported by the preinstalled Text-to-speech engine) accessibility services as provided in the talkback open source project .

If Television device implementations report the feature android.hardware.audio.output , they:

- [3.11 /T-SR] Are STRONGLY RECOMMENDED to include a TTS engine supporting the languages available on the device.
- [3.11 /T-1-1] MUST support installation of third-party TTS engines.

Television device implementations:

- [3.12 /T-0-1] MUST support TV Input Framework.

2.3.4. Performance and Power

- [8.1 /T-0-1] **Consistent frame latency** . Inconsistent frame latency or a delay to render frames MUST NOT happen more often than 5 frames in a second, and SHOULD be below 1 frames in a second.
- [8.2 /T-0-1] MUST ensure a sequential write performance of at least 5MB/s.
- [8.2 /T-0-2] MUST ensure a random write performance of at least 0.5MB/s.
- [8.2 /T-0-3] MUST ensure a sequential read performance of at least 15MB/s.
- [8.2 /T-0-4] MUST ensure a random read performance of at least 3.5MB/s.

If Television device implementations include features to improve device power management that are included in AOSP or extend the features that are included in AOSP, they:

- [8.3 /T-1-1] MUST provide user affordance to enable and disable the battery saver feature.

If Television device implementations do not have a battery they:

- [8.3 /T-1-2] MUST register the device as a batteryless device as described inSupporting Batteryless Devices .

If Television device implementations have a battery they:

- [8.3 /T-1-3] MUST provide user affordance to display all apps that are exempted from App Standby and Doze power-saving modes.

Television device implementations:

- [8.4 /T-0-1] MUST provide a per-component power profile that defines the current consumption value for each hardware component and the approximate battery drain caused by the components over time as documented in the Android Open Source Project site.
- [8.4 /T-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [8.4 /T-0-3] MUST report CPU power consumption per each process's UID. The Android Open Source Project meets the requirement through the uid_cputime kernel module implementation.
- [8.4 /T] SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.
- [8.4 /T-0-4] MUST make this power usage available via the adb shell dumpsys batterystats

shell command to the app developer.

2.3.5. Security Model

Television device implementations:

- [9.11 /T-0-1] MUST back up the keystore implementation with an isolated execution environment.
- [9.11 /T-0-2] MUST have implementations of RSA, AES, ECDSA and HMAC cryptographic algorithms and MD5, SHA1, and SHA-2 family hash functions to properly support the Android Keystore system's supported algorithms in an area that is securely isolated from the code running on the kernel and above. Secure isolation MUST block all potential mechanisms by which kernel or userspace code might access the internal state of the isolated environment, including DMA. The upstream Android Open Source Project (AOSP) meets this requirement by using the Trusty implementation, but another ARM TrustZone-based solution or a third-party reviewed secure implementation of a proper hypervisor-based isolation are alternative options.
- [9.11 /T-0-3] MUST perform the lock screen authentication in the isolated execution environment and only when successful, allow the authentication-bound keys to be used. Lock screen credentials MUST be stored in a way that allows only the isolated execution environment to perform lock screen authentication. The upstream Android Open Source Project provides the Gatekeeper Hardware Abstraction Layer (HAL) and Trusty, which can be used to satisfy this requirement.
- [9.11 /T-0-4] MUST support key attestation where the attestation signing key is protected by secure hardware and signing is performed in secure hardware. The attestation signing keys MUST be shared across large enough number of devices to prevent the keys from being used as device identifiers. One way of meeting this requirement is to share the same attestation key unless at least 100,000 units of a given SKU are produced. If more than 100,000 units of an SKU are produced, a different key MAY be used for each 100,000 units.

Note that if a device implementation is already launched on an earlier Android version, such a device is exempted from the requirement to have a keystore backed by an isolated execution environment and support the key attestation, unless it declares the android.hardware.fingerprint feature which requires a keystore backed by an isolated execution environment.

If Television device implementations support a secure lock screen, they:

- [9.11 /T-1-1] MUST allow the user to choose the Sleep timeout for transition from the unlocked to the locked state, with a minimum allowable timeout up to 15 seconds or less.

2.3.6. Developer Tools and Options Compatibility

Television device implementations:

- **Perfetto**
  - [6.1 /T-0-1] MUST expose a /system/bin/perfetto binary to the shell user which cmdline complies with the perfetto documentation .
  - [6.1 /T-0-2] The perfetto binary MUST accept as input a protobuf config that complies with the schema defined in the perfetto documentation .
  - [6.1 /T-0-3] The perfetto binary MUST write as output a protobuf trace that complies with the schema defined in the perfetto documentation .
  - [6.1 /T-0-4] MUST provide, through the perfetto binary, at least the data sources described in the perfetto documentation .

2.4. Watch Requirements

An **Android Watch device** refers to an Android device implementation intended to be worn on the body, perhaps on the wrist.

Android device implementations are classified as a Watch if they meet all the following criteria:

- Have a screen with the physical diagonal length in the range from 1.1 to 2.5 inches.
- Have a mechanism provided to be worn on the body.

The additional requirements in the rest of this section are specific to Android Watch device implementations.

2.4.1. Hardware

Watch device implementations:

- [7.1 .1.1/W-0-1] MUST have a screen with the physical diagonal size in the range from 1.1 to 2.5 inches.
- [7.2 .3/W-0-1] MUST have the Home function available to the user, and the Back function except for when it is in UI_MODE_TYPE_WATCH .
- [7.2 .4/W-0-1] MUST support touchscreen input.
- [7.3 .1/W-SR] Are STRONGLY RECOMMENDED to include a 3-axis accelerometer.

If Watch device implementations include a GPS/GNSS receiver and report the capability to applications through the android.hardware.location.gps feature flag, they:

- [7.3 .3/W-1-1] MUST report GNSS measurements, as soon as they are found, even if a location calculated from GPS/GNSS is not yet reported.
- [7.3 .3/W-1-2] MUST report GNSS pseudoranges and pseudorange rates, that, in open-sky conditions after determining the location, while stationary or moving with less than 0.2 meter per second squared of acceleration, are sufficient to calculate position within 20 meters, and speed within 0.2 meters per second, at least 95% of the time.

If Watch device implementations include a 3-axis gyroscope, they:

- [7.3 .4/W-2-1] MUST be capable of measuring orientation changes up to 1000 degrees per second.

Watch device implementations:

- [7.4 .3/W-0-1] MUST support Bluetooth.
- [7.6 .1/W-0-1] MUST have at least 1 GB of non-volatile storage available for application private data (a.k.a. "/data" partition).
- [7.6 .1/W-0-2] MUST have at least 416 MB memory available to the kernel and userspace.
- [7.8 .1/W-0-1] MUST include a microphone.
- [7.8 .2/W] MAY have audio output.

2.4.2. Multimedia

No additional requirements.

2.4.3. Software

Watch device implementations:

- [3 /W-0-1] MUST declare the feature android.hardware.type.watch .
- [3 /W-0-2] MUST support uiMode = UI_MODE_TYPE_WATCH .
- [3.2.3.1 /W-0-1] MUST preload one or more applications or service components with an intent handler, for all the public intent filter patterns defined by the following application intents listed here .

Watch device implementations:

- [3.8 .4/W-SR] Are STRONGLY RECOMMENDED to implement an assistant on the device to handle the Assist action .

Watch device implementations that declare the android.hardware.audio.output feature flag:

- [3.10 /W-1-1] MUST support third-party accessibility services.
- [3.10 /W-SR] Are STRONGLY RECOMMENDED to preload accessibility services on the device comparable with or exceeding functionality of the Switch Access and TalkBack (for languages supported by the preinstalled Text-to-speech engine) accessibility services as provided in the talkback open source project .

If Watch device implementations report the feature android.hardware.audio.output, they:

- [3.11 /W-SR] Are STRONGLY RECOMMENDED to include a TTS engine supporting the languages available on the device.
- [3.11 /W-0-1] MUST support installation of third-party TTS engines.

2.4.4. Performance and Power

If Watch device implementations include features to improve device power management that are included in AOSP or extend the features that are included in AOSP, they:

- [8.3 /W-SR] Are STRONGLY RECOMMENDED to provide user affordance to display all apps that are exempted from App Standby and Doze power-saving modes.
- [8.3 /W-SR] Are STRONGLY RECOMMENDED to provide user affordance to enable and disable the battery saver feature.

Watch device implementations:

- [8.4 /W-0-1] MUST provide a per-component power profile that defines the current consumption value for each hardware component and the approximate battery drain caused by the components over time as documented in the Android Open Source Project site.
- [8.4 /W-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [8.4 /W-0-3] MUST report CPU power consumption per each process's UID. The Android Open Source Project meets the requirement through the uid_cputime kernel module implementation.
- [8.4 /W-0-4] MUST make this power usage available via the adb shell dumpsys batterystats shell command to the app developer.
- [8.4 /W] SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.

2.5. Automotive Requirements

Android Automotive implementation refers to a vehicle head unit running Android as an operating system for part or all of the system and/or infotainment functionality.

Android device implementations are classified as an Automotive if they declare the feature android.hardware.type.automotive or meet all the following criteria.

- Are embedded as part of, or pluggable to, an automotive vehicle.
- Are using a screen in the driver's seat row as the primary display.

The additional requirements in the rest of this section are specific to Android Automotive device implementations.

2.5.1. Hardware

Automotive device implementations:

- [7.1 .1.1/A-0-1] MUST have a screen at least 6 inches in physical diagonal size.
- [7.1 .1.1/A-0-2] MUST have a screen size layout of at least 750 dp x 480 dp.
- [7.2 .3/A-0-1] MUST provide the Home function and MAY provide Back and Recent functions.
- [7.2 .3/A-0-2] MUST send both the normal and long press event of the Back function (KEYCODE_BACK) to the foreground application.
- [7.3 /A-0-1] MUST implement and report GEAR_SELECTION , NIGHT_MODE , PERF_VEHICLE_SPEED and PARKING_BRAKE_ON .
- [7.3 /A-0-2] The value of the NIGHT_MODE flag MUST be consistent with dashboard day/night mode and SHOULD be based on ambient light sensor input. The underlying ambient light sensor MAY be the same as Photometer .
- [7.3 /A-0-3] MUST provide sensor additional info field TYPE_SENSOR_PLACEMENT as part of SensorAdditionalInfo for every sensor provided.
- [7.3 /A-0-1] MAY dead reckon Location by fusing GPS/GNSS with additional sensors. If Location is dead reckoned, it is STRONGLY RECOMMENDED to implement and report the corresponding Sensor types and/or Vehicle Property IDs used.
- [7.3 /A-0-2] The Location requested via LocationManager#requestLocationUpdates()

MUST NOT be map matched.

If Automotive device implementations include a 3-axis accelerometer, they:

- [7.3 .1/A-1-1] MUST be able to report events up to a frequency of at least 100 Hz.
- [7.3 .1/A-1-2] MUST comply with the Android car sensor coordinate system .

If Automotive device implementations include a 3-axis gyroscope, they:

- [7.3 .4/A-2-1] MUST be able to report events up to a frequency of at least 100 Hz.
- [7.3 .4/A-2-2] MUST also implement the TYPE_GYROSCOPE_UNCALIBRATED sensor.
- [7.3 .4/A-2-3] MUST be capable of measuring orientation changes up to 250 degrees per second.
- [7.3 .4/A-SR] Are STRONGLY RECOMMENDED to configure the gyroscope's measurement range to +/-250dps in order to maximize the resolution possible

If Automotive device implementations include a GPS/GNSS receiver, but do not include cellular network-based data connectivity, they:

- [7.3 .3/A-3-1] MUST determine location the very first time the GPS/GNSS receiver is turned on or after 4+ days within 60 seconds.
- [7.3 .3/A-3-2] MUST meet the time-to-first-fix criteria as described in 7.3.3/C-1-2 and 7.3.3/C-1-6 for all other location requests (i.e requests which are not the first time ever or after 4+ days). The requirement 7.3.3/C-1-2 is typically met in vehicles without cellular network-based data connectivity, by using GNSS orbit predictions calculated on the receiver, or using the last known vehicle location along with the ability to dead reckon for at least 60 seconds with a position accuracy satisfying 7.3.3/C-1-3 , or a combination of both.

Automotive device implementations:

- [7.4 .3/A-0-1] MUST support Bluetooth and SHOULD support Bluetooth LE.
- [7.4 .3/A-0-2] Android Automotive implementations MUST support the following Bluetooth profiles:
  - Phone calling over Hands-Free Profile (HFP).
  - Media playback over Audio Distribution Profile (A2DP).
  - Media playback control over Remote Control Profile (AVRCP).
  - Contact sharing using the Phone Book Access Profile (PBAP).
- [7.4 .3/A-SR] Are STRONGLY RECOMMENDED to support Message Access Profile (MAP).
- [7.4 .5/A] SHOULD include support for cellular network-based data connectivity.
- [7.4 .5/A] MAY use the System API NetworkCapabilities#NET_CAPABILITY_OEM_PAID constant for networks that should be available to system apps.

An exterior view camera is a camera that images scenes outside of the device implementation, like a dashcam.

Automotive device implementations:

- SHOULD include one or more exterior view cameras.

If Automotive device implementations include an exterior view camera, for such a camera, they:

- [7.5 /A-1-1] MUST NOT have exterior view cameras accessible via the Android Camera APIs , unless they comply with camera core requirements .
- [7.5 /A-SR] Are STRONGLY RECOMMENDED not to rotate or horizontally mirror the camera preview.
- [7.5 .5/A-SR] Are STRONGLY RECOMMENDED to be oriented so that the long dimension of the camera aligns with the horizon.
- [7.5 /A-SR] Are STRONGLY RECOMMENDED to have a resolution of at least 1.3 megapixels.
- SHOULD have either fixed-focus or EDOF (extended depth of field) hardware.
- SHOULD support Android Synchronization Framework .
- MAY have either hardware auto-focus or software auto-focus implemented in the camera driver.

Automotive device implementations:

- [7.6 .1/A-0-1] MUST have at least 4 GB of non-volatile storage available for application private data (a.k.a. "/data" partition).
- [7.6 .1/A] SHOULD format the data partition to offer improved performance and longevity on flash storage, for example using f2fs file-system.

If Automotive device implementations provide shared external storage via a portion of the internal non-removable storage, they:

- [7.6 .1/A-SR] Are STRONGLY RECOMMENDED to reduce I/O overhead on operations performed on the external storage, for example by using SDCardFS .

If Automotive device implementations are 32-bit:

- [7.6 .1/A-1-1] The memory available to the kernel and userspace MUST be at least 512MB if any of the following densities are used:
  - 280dpi or lower on small/normal screens
  - ldpi or lower on extra large screens
  - mdpi or lower on large screens
- [7.6 .1/A-1-2] The memory available to the kernel and userspace MUST be at least 608MB if any of the following densities are used:
  - xhdpi or higher on small/normal screens
  - hdpi or higher on large screens
  - mdpi or higher on extra large screens
- [7.6 .1/A-1-3] The memory available to the kernel and userspace MUST be at least 896MB if any of the following densities are used:
  - 400dpi or higher on small/normal screens
  - xhdpi or higher on large screens
  - tvdpi or higher on extra large screens
- [7.6 .1/A-1-4] The memory available to the kernel and userspace MUST be at least 1344MB if any of the following densities are used:
  - 560dpi or higher on small/normal screens
  - 400dpi or higher on large screens
  - xhdpi or higher on extra large screens

If Automotive device implementations are 64-bit:

- [7.6 .1/A-2-1] The memory available to the kernel and userspace MUST be at least 816MB if any of the following densities are used:
  - 280dpi or lower on small/normal screens
  - ldpi or lower on extra large screens
  - mdpi or lower on large screens
- [7.6 .1/A-2-2] The memory available to the kernel and userspace MUST be at least 944MB if any of the following densities are used:
  - xhdpi or higher on small/normal screens
  - hdpi or higher on large screens
  - mdpi or higher on extra large screens
- [7.6 .1/A-2-3] The memory available to the kernel and userspace MUST be at least 1280MB if any of the following densities are used:
  - 400dpi or higher on small/normal screens
  - xhdpi or higher on large screens
  - tvdpi or higher on extra large screens
- [7.6 .1/A-2-4] The memory available to the kernel and userspace MUST be at least 1824MB if any of the following densities are used:
  - 560dpi or higher on small/normal screens
  - 400dpi or higher on large screens
  - xhdpi or higher on extra large screens

Note that the "memory available to the kernel and userspace" above refers to the memory space

provided in addition to any memory already dedicated to hardware components such as radio, video, and so on that are not under the kernel's control on device implementations.

Automotive device implementations:

- [7.7 .1/A] SHOULD include a USB port supporting peripheral mode.

Automotive device implementations:

- [7.8 .1/A-0-1] MUST include a microphone.

Automotive device implementations:

- [7.8 .2/A-0-1] MUST have an audio output and declare android.hardware.audio.output .

2.5.2. Multimedia

Automotive device implementations MUST support the following audio encoding and decoding formats and make them available to third-party applications:

- [5.1 /A-0-1] MPEG-4 AAC Profile (AAC LC)
- [5.1 /A-0-2] MPEG-4 HE AAC Profile (AAC+)
- [5.1 /A-0-3] AAC ELD (enhanced low delay AAC)

Automotive device implementations MUST support the following video encoding formats and make them available to third-party applications:

- [5.2 /A-0-1] H.264 AVC
- [5.2 /A-0-2] VP8

Automotive device implementations MUST support the following video decoding formats and make them available to third-party applications:

- [5.3 /A-0-1] H.264 AVC
- [5.3 /A-0-2] MPEG-4 SP
- [5.3 /A-0-3] VP8
- [5.3 /A-0-4] VP9

Automotive device implementations are STRONGLY RECOMMENDED to support the following video decoding:

- [5.3 /A-SR] H.265 HEVC

2.5.3. Software

Automotive device implementations:

- [3 /A-0-1] MUST declare the feature android.hardware.type.automotive .
- [3 /A-0-2] MUST support uiMode = UI_MODE_TYPE_CAR .
- [3 /A-0-3] MUST support all public APIs in the android.car.* namespace.

If Automotive device implementations provide a proprietary API using android.car.CarPropertyManager with android.car.VehiclePropertyIds , they:

- [3 /A-1-1] MUST NOT attach special privileges to system application's use of these properties, or prevent third-party applications from using these properties.
- [3 /A-1-2] MUST NOT replicate a vehicle property that already exists in the SDK .

Automotive device implementations:

- [3.2 .1/A-0-1] MUST support and enforce all permissions constants as documented by the Automotive Permission reference page .
- [3.2.3.1 /A-0-1] MUST preload one or more applications or service components with an intent handler, for all the public intent filter patterns defined by the following application intents listed here .

- [3.4 .1/A-0-1] MUST provide a complete implementation of the android.webkit.Webview API.
- [3.8 .3/A-0-1] MUST display notifications that use the Notification.CarExtender API when requested by third-party applications.
- [3.8 .4/A-SR] Are Strongly Recommended to implement an assistant on the device to handle the Assist action .

If Automotive device implementations include a push-to-talk button, they:

- [3.8 .4/A-1-1] MUST use a short press of the push-to-talk button as the designated interaction to launch the user-selected assist app, in other words the app that implements VoiceInteractionService .

Automotive device implementations:

- [3.8.3.1 /A-0-1] MUST correctly render resources as described in the Notifications on Automotive OS SDK documentation.
- [3.8.3.1 /A-0-2] MUST display PLAY and MUTE for notification actions in the place of those provided through Notification.Builder.addAction()
- [3.8.3.1 /A] SHOULD restrict the use of rich management tasks such as per-notification-channel controls. MAY use UI affordance per application to reduce controls.

Automotive device implementations:

- [3.14 /A-0-1] MUST include a UI framework to support third-party apps using the media APIs as described in section 3.14 .
- [3.14 /A-0-2] MUST allow the user to safely interact with Media Applications while driving.
- [3.14 /A-0-3] MUST support the CAR_INTENT_ACTION_MEDIA_TEMPLATE implicit Intent action with the CAR_EXTRA_MEDIA_PACKAGE extra.
- [3.14 /A-0-4] MUST provide an affordance to navigate into a Media Application's preference activity , but MUST only enable it when Car UX Restrictions are not in effect.
- [3.14 /A-0-5] MUST display error messages set by Media Applications, and MUST support the optional extras ERROR_RESOLUTION_ACTION_LABEL and ERROR_RESOLUTION_ACTION_INTENT .
- [3.14 /A-0-6] MUST support an in-app search affordance for apps that support searching.
- [3.14 /A-0-7] MUST respect CONTENT_STYLE_BROWSABLE_HINT and CONTENT_STYLE_PLAYABLE_HINT definitions when displaying the MediaBrowser hierarchy.

If Automotive device implementations include a default launcher app, they:

- [3.14 /A-1-1] MUST include media services and open them with the CAR_INTENT_ACTION_MEDIA_TEMPLATE intent.

Automotive device implementations:

- [3.8 /A] MAY restrict the application requests to enter a full screen mode as described in immersive documentation .
- [3.8 /A] MAY keep the status bar and the navigation bar visible at all times.
- [3.8 /A] MAY restrict the application requests to change the colors behind the system UI elements, to ensure those elements are clearly visible at all times.

2.5.4. Performance and Power

Automotive device implementations:

- [8.2 /A-0-1] MUST report the number of bytes read and written to non-volatile storage per each process's UID so the stats are available to developers through System API android.car.storagemonitoring.CarStorageMonitoringManager . The Android Open Source Project meets the requirement through the uid_sys_stats kernel module.
- [8.3 /A-1-3] MUST support Garage Mode .
- [8.3 /A] SHOULD be in Garage Mode for at least 15 minutes after every drive unless:
  - The battery is drained.

    - No idle jobs are scheduled.
    - The driver exits Garage Mode.
- [8.4 /A-0-1] MUST provide a per-component power profile that defines the current consumption value for each hardware component and the approximate battery drain caused by the components over time as documented in the Android Open Source Project site.
- [8.4 /A-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [8.4 /A-0-3] MUST report CPU power consumption per each process's UID. The Android Open Source Project meets the requirement through the uid_cputime kernel module implementation.
- [8.4 /A] SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.
- [8.4 /A-0-4] MUST make this power usage available via the adb shell dumpsys batterystats shell command to the app developer.

2.5.5. Security Model

If Automotive device implementations support multiple users, they:

- [9.5 /A-1-1] MUST NOT allow users to interact with nor switch into the Headless System User , except for device provisioning .
- [9.5 /A-1-2] MUST switch into a Secondary User before BOOT_COMPLETED .
- [9.5 /A-1-3] MUST support the ability to create a Guest User even when the maximum number of Users on a device has been reached.

Automotive device implementations:

- [9.11 /A-0-1] MUST back up the keystore implementation with an isolated execution environment.
- [9.11 /A-0-2] MUST have implementations of RSA, AES, ECDSA and HMAC cryptographic algorithms and MD5, SHA1, and SHA-2 family hash functions to properly support the Android Keystore system's supported algorithms in an area that is securely isolated from the code running on the kernel and above. Secure isolation MUST block all potential mechanisms by which kernel or userspace code might access the internal state of the isolated environment, including DMA. The upstream Android Open Source Project (AOSP) meets this requirement by using the Trusty implementation, but another ARM TrustZone-based solution or a third-party reviewed secure implementation of a proper hypervisor-based isolation are alternative options.
- [9.11 /A-0-3] MUST perform the lock screen authentication in the isolated execution environment and only when successful, allow the authentication-bound keys to be used. Lock screen credentials MUST be stored in a way that allows only the isolated execution environment to perform lock screen authentication. The upstream Android Open Source Project provides the Gatekeeper Hardware Abstraction Layer (HAL) and Trusty, which can be used to satisfy this requirement.
- [9.11 /A-0-4] MUST support key attestation where the attestation signing key is protected by secure hardware and signing is performed in secure hardware. The attestation signing keys MUST be shared across large enough number of devices to prevent the keys from being used as device identifiers. One way of meeting this requirement is to share the same attestation key unless at least 100,000 units of a given SKU are produced. If more than 100,000 units of an SKU are produced, a different key MAY be used for each 100,000 units.

Note that if a device implementation is already launched on an earlier Android version, such a device is exempted from the requirement to have a keystore backed by an isolated execution environment and support the key attestation, unless it declares the android.hardware.fingerprint feature which requires a keystore backed by an isolated execution environment.

Automotive device implementations:

- [9.14 /A-0-1] MUST gatekeep messages from Android framework vehicle subsystems, e.g., whitelisting permitted message types and message sources.
- [9.14 /A-0-2] MUST watchdog against denial of service attacks from the Android framework or third-party apps. This guards against malicious software flooding the vehicle network with traffic, which may lead to malfunctioning vehicle subsystems.

2.5.6. Developer Tools and Options Compatibility

Automotive device implementations:

- Perfetto
  - [6.1 /A-0-1] MUST expose a /system/bin/perfetto binary to the shell user which cmdline complies with the perfetto documentation .
  - [6.1 /A-0-2] The perfetto binary MUST accept as input a protobuf config that complies with the schema defined in the perfetto documentation .
  - [6.1 /A-0-3] The perfetto binary MUST write as output a protobuf trace that complies with the schema defined in the perfetto documentation .
  - [6.1 /A-0-4] MUST provide, through the perfetto binary, at least the data sources described in the perfetto documentation .

2.6. Tablet Requirements

An **Android Tablet device** refers to an Android device implementation that typically meets all the following criteria:

- Used by holding in both hands.
- Does not have a clamshell or convertible configuration.
- Physical keyboard implementations used with the device connect by means of a standard connection (e.g. USB, Bluetooth).
- Has a power source that provides mobility, such as a battery.

Tablet device implementations have similar requirements to handheld device implementations. The exceptions are indicated by an * in that section and noted for reference in this section.

2.6.1. Hardware

Screen Size

- [7.1 .1.1/Tab-0-1] MUST have a screen in the range of 7 to 18 inches.

Gyroscope

If Tablet device implementations include a 3-axis gyroscope, they:

- [7.3 .4/Tab-1-1] MUST be capable of measuring orientation changes up to 1000 degrees per second.

Minimum Memory and Storage (Section 7.6.1)

The screen densities listed for small/normal screens in the handheld requirements are not applicable to tablets.

USB peripheral mode (Section 7.7.1)

If tablet device implementations include a USB port supporting peripheral mode, they:

- [7.7.1 /Tab] MAY implement the Android Open Accessory (AOA) API.

Virtual Reality Mode (Section 7.9.1)

Virtual Reality High Performance (Section 7.9.2)

Virtual reality requirements are not applicable to tablets.

Keys and Credentials (Section 9.11)

Refer to Section [9.11].

2.6.2. Software

- [3.2.3.1 /Tab-0-1] MUST preload one or more applications or service components with an intent handler, for all the public intent filter patterns defined by the following application intents listed here .

3. Software

3.1. Managed API Compatibility

The managed Dalvik bytecode execution environment is the primary vehicle for Android applications. The Android application programming interface (API) is the set of Android platform interfaces exposed to applications running in the managed runtime environment.

Device implementations:

- [C-0-1] MUST provide complete implementations, including all documented behaviors, of any documented API exposed by the Android SDK or any API decorated with the "@SystemApi" marker in the upstream Android source code.
- [C-0-2] MUST support/preserve all classes, methods, and associated elements marked by the TestApi annotation (@TestApi).
- [C-0-3] MUST NOT omit any managed APIs, alter API interfaces or signatures, deviate from the documented behavior, or include no-ops, except where specifically allowed by this Compatibility Definition.
- [C-0-4] MUST still keep the APIs present and behave in a reasonable way, even when some hardware features for which Android includes APIs are omitted. See section 7 for specific requirements for this scenario.
- [C-0-5] MUST NOT allow third-party apps to use non-SDK interfaces, which are defined as methods and fields in the Java language packages that are in the boot classpath in AOSP, and that do not form part of the public SDK. This includes APIs decorated with the `@hide` annotation but not with a `@SystemAPI` , as described in the SDK documents and private and package-private class members.
- [C-0-6] MUST ship with each and every non-SDK interface on the same restricted lists as provided via the `greylist` , `greylist-max-o` , `greylist-max-p` , and blacklist flags in `prebuilts/runtime/appcompat/hiddenapi-flags.csv` path for the appropriate API level branch in the AOSP.
- [C-0-7] MUST support the signed config dynamic update mechanism to remove non-SDK interfaces from a restricted list by embedding signed configuration in any APK, using the existing public keys present in AOSP.

  However they:
  - MAY, if a hidden API is absent or implemented differently on the device implementation, move the hidden API into the blacklist or omit it from all restricted lists (i.e. light-grey, dark-grey, black).
  - MAY, if a hidden API does not already exist in the AOSP, add the hidden API to any of the restricted lists (i.e. light-grey, dark-grey, black).

3.1.1. Android Extensions

Android supports extending the managed API surface of a particular API level by updating the extension version for that API level. The `android.os.ext.SdkExtensions.getExtensionVersion(int apiLevel)` API returns the extension version of the provided `apiLevel` , if there are extensions for that API level.

Android device implementations:

- [C-0-1] MUST preload the AOSP implementation of both the shared library `ExtShared` and services `ExtServices` with versions greater than or equal to the minimum versions allowed per each API level. For example, Android 7.0 device implementations, running API level 24 MUST include at least version 1.
- [C-0-2] MUST only return valid extension version number that have been defined by the AOSP.
- [C-0-3] MUST support all the APIs defined by the extension versions returned by `android.os.ext.SdkExtensions.getExtensionVersion(int apiLevel)` in the same manner as other managed APIs are supported, following the requirements in section 3.1 .

3.1.2. Android Library

Due to Apache HTTP client deprecation , device implementations:

- [C-0-1] MUST NOT place the `org.apache.http.legacy` library in the bootclasspath.
- [C-0-2] MUST add the `org.apache.http.legacy` library to the application classpath only when the app satisfies one of the following conditions:
  - Targets API level 28 or lower.
  - Declares in its manifest that it needs the library by setting the `android:name` attribute of `<uses-library>` to `org.apache.http.legacy` .

The AOSP implementation meets these requirements.

3.2. Soft API Compatibility

In addition to the managed APIs from section 3.1 , Android also includes a significant runtime-only "soft" API, in the form of such things as intents, permissions, and similar aspects of Android applications that cannot be enforced at application compile time.

3.2.1. Permissions

- [C-0-1] Device implementers MUST support and enforce all permission constants as documented by the Permission reference page . Note that section 9 lists additional requirements related to the Android security model.

3.2.2. Build Parameters

The Android APIs include a number of constants on the android.os.Build class that are intended to describe the current device.

- [C-0-1] To provide consistent, meaningful values across device implementations, the table below includes additional restrictions on the formats of these values to which device implementations MUST conform.

Parameter	Details
VERSION.RELEASE	The version of the currently-executing Android system, in human-readable format. This field MUST have one of the string values defined in 11 .
VERSION.SDK	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 11, this field MUST have the integer value 11_INT.
VERSION.SDK_INT	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 11, this field MUST have the integer value 11_INT.
VERSION.INCREMENTAL	A value chosen by the device implementer designating the specific build of the currently-executing Android system, in human-readable format. This value MUST NOT be reused for different builds made available to end users. A typical use of this field is to indicate which build number or source-control change identifier was used to generate the build. The value of this field MUST be encodable as printable 7-bit ASCII and match the regular expression "^[^ :\/~]+$".
BOARD	A value chosen by the device implementer identifying the specific internal hardware used by the device, in human-readable format. A possible use of this field is to indicate the specific revision of the board powering the device. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "^[a-zA-Z0-9_-]+$".
BRAND	A value reflecting the brand name associated with the device as known to the end users. MUST be in human-readable format and SHOULD represent the manufacturer of the device or the company brand under which the device is marketed. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "^[a-zA-Z0-9_-]+$".
SUPPORTED_ABIS	The name of the instruction set (CPU type + ABI convention) of native code. See section 3.3. Native API Compatibility .
SUPPORTED_32_BIT_ABIS	The name of the instruction set (CPU type + ABI convention) of native code. See section 3.3. Native API Compatibility .
SUPPORTED_64_BIT_ABIS	The name of the second instruction set (CPU type + ABI convention) of native code. See section 3.3. Native API Compatibility .
CPU_ABI	The name of the instruction set (CPU type + ABI convention) of native code. See section 3.3. Native API Compatibility .
CPU_ABI2	The name of the second instruction set (CPU type + ABI convention) of native code. See section 3.3. Native API Compatibility .

DEVICE	A value chosen by the device implementer containing the development name or code name identifying the configuration of the hardware features and industrial design of the device. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9_-]+$”. This device name MUST NOT change during the lifetime of the product.

FINGERPRINT	A string that uniquely identifies this build. It SHOULD be reasonably human-readable. It MUST follow this template:
	`$(BRAND)/$(PRODUCT)/`
	`$(DEVICE):$(VERSION.RELEASE)/$(ID)/$(VERSION.INCREMENTAL):$(TYPE)/$(TAGS)`
	For example:
	`acme/myproduct/`
	`mydevice:11/LMYXX/3359:userdebug/test-keys`
	The fingerprint MUST NOT include whitespace characters. The value of this field MUST be encodable as 7-bit ASCII.

HARDWARE	The name of the hardware (from the kernel command line or /proc). It SHOULD be reasonably human-readable. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9_-]+$”.
HOST	A string that uniquely identifies the host the build was built on, in human-readable format. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
ID	An identifier chosen by the device implementer to refer to a specific release, in human-readable format. This field can be the same as android.os.Build.VERSION.INCREMENTAL, but SHOULD be a value sufficiently meaningful for end users to distinguish between software builds. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9._-]+$”.
MANUFACTURER	The trade name of the Original Equipment Manufacturer (OEM) of the product. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string (""). This field MUST NOT change during the lifetime of the product.
MODEL	A value chosen by the device implementer containing the name of the device as known to the end user. This SHOULD be the same name under which the device is marketed and sold to end users. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string (""). This field MUST NOT change during the lifetime of the product.
PRODUCT	A value chosen by the device implementer containing the development name or code name of the specific product (SKU) that MUST be unique within the same brand. MUST be human-readable, but is not necessarily intended for view by end users. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9_-]+$”. This product name MUST NOT change during the lifetime of the product.
SERIAL	MUST return "UNKNOWN".
TAGS	A comma-separated list of tags chosen by the device implementer that further distinguishes the build. The tags MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9._-]+” and MUST have one of the values corresponding to the three typical Android platform signing configurations: release-keys, dev-keys, and test-keys.
TIME	A value representing the timestamp of when the build occurred.
TYPE	A value chosen by the device implementer specifying the runtime configuration of the build. This field MUST have one of the values corresponding to the three typical Android runtime configurations: user, userdebug, or eng.
USER	A name or user ID of the user (or automated user) that generated the build. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").

SECURITY_PATCH	A value indicating the security patch level of a build. It MUST signify that the build is not in any way vulnerable to any of the issues described up through the designated Android Public Security Bulletin. It MUST be in the format [YYYY-MM-DD], matching a defined string documented in the Android Public Security Bulletin or in the Android Security Advisory , for example "2015-11-01".
BASE_OS	A value representing the FINGERPRINT parameter of the build that is otherwise identical to this build except for the patches provided in the Android Public Security Bulletin. It MUST report the correct value and if such a build does not exist, report an empty string ("").
BOOTLOADER	A value chosen by the device implementer identifying the specific internal bootloader version used in the device, in human-readable format. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9._-]+$”.
getRadioVersion()	MUST (be or return) a value chosen by the device implementer identifying the specific internal radio/modem version used in the device, in human-readable format. If a device does not have any internal radio/modem it MUST return NULL. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9._-,]+$”.
getSerial()	MUST (be or return) a hardware serial number, which MUST be available and unique across devices with the same MODEL and MANUFACTURER. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9._-,]+$”.

3.2.3. Intent Compatibility

3.2.3.1. Common Application Intents

Android intents allow application components to request functionality from other Android components. The Android upstream project includes a list of applications which implement several intent patterns to perform common actions.

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to preload one or more applications or service components with an intent handler, for all the public intent filter patterns defined by the following application intents listed here and provide fulfillment i.e meet with the developer expectation for these common application intents as described in the SDK.

Please refer to Section 2 for mandatory application intents for each device type.

3.2.3.2. Intent Resolution

- [C-0-1] As Android is an extensible platform, device implementations MUST allow each intent pattern referenced in section 3.2.3.1 , except for Settings, to be overridden by third-party applications. The upstream Android open source implementation allows this by default.
- [C-0-2] Device implementers MUST NOT attach special privileges to system applications' use of these intent patterns, or prevent third-party applications from binding to and assuming control of these patterns. This prohibition specifically includes but is not limited to disabling the “Chooser” user interface that allows the user to select between multiple applications that all handle the same intent pattern.
- [C-0-3] Device implementations MUST provide a user interface for users to modify the default activity for intents.
- However, device implementations MAY provide default activities for specific URI patterns (e.g. http://play.google.com) when the default activity provides a more specific attribute for the data URI. For example, an intent filter pattern specifying the data URI “http://www.android.com” is more specific than the browser's core intent pattern for “http://”.

Android also includes a mechanism for third-party apps to declare an authoritative default app linking behavior for certain types of web URI intents. When such authoritative declarations are defined in an

app's intent filter patterns, device implementations:

- [C-0-4] MUST attempt to validate any intent filters by performing the validation steps defined in the Digital Asset Links specification as implemented by the Package Manager in the upstream Android Open Source Project.
- [C-0-5] MUST attempt validation of the intent filters during the installation of the application and set all successfully validated URI intent filters as default app handlers for their URIs.
- MAY set specific URI intent filters as default app handlers for their URIs, if they are successfully verified but other candidate URI filters fail verification. If a device implementation does this, it MUST provide the user appropriate per-URI pattern overrides in the settings menu.
- MUST provide the user with per-app App Links controls in Settings as follows:
  - [C-0-6] The user MUST be able to override holistically the default app links behavior for an app to be: always open, always ask, or never open, which must apply to all candidate URI intent filters equally.
  - [C-0-7] The user MUST be able to see a list of the candidate URI intent filters.
  - The device implementation MAY provide the user with the ability to override specific candidate URI intent filters that were successfully verified, on a per-intent filter basis.
  - [C-0-8] The device implementation MUST provide users with the ability to view and override specific candidate URI intent filters if the device implementation lets some candidate URI intent filters succeed verification while some others can fail.

3.2.3.3. Intent Namespaces

- [C-0-1] Device implementations MUST NOT include any Android component that honors any new intent or broadcast intent patterns using an ACTION, CATEGORY, or other key string in the android. *or com.android.* namespace.
- [C-0-2] Device implementers MUST NOT include any Android components that honor any new intent or broadcast intent patterns using an ACTION, CATEGORY, or other key string in a package space belonging to another organization.
- [C-0-3] Device implementers MUST NOT alter or extend any of the intent patterns listed in section 3.2.3.1 .
- Device implementations MAY include intent patterns using namespaces clearly and obviously associated with their own organization. This prohibition is analogous to that specified for Java language classes in section 3.6 .

3.2.3.4. Broadcast Intents

Third-party applications rely on the platform to broadcast certain intents to notify them of changes in the hardware or software environment.

Device implementations:

- [C-0-1] MUST broadcast the public broadcast intents listed here in response to appropriate system events as described in the SDK documentation. Note that this requirement is not conflicting with section 3.5 as the limitation for background applications are also described in the SDK documentation. Also certain broadcast intents are conditional upon hardware support, if the device supports the necessary hardware they MUST broadcast the intents and provide the behavior inline with SDK documentation.

3.2.3.5. Conditional Application Intents

Android includes settings that provide users an easy way to select their default applications, for example for Home screen or SMS.

Where it makes sense, device implementations MUST provide a similar settings menu and be compatible with the intent filter pattern and API methods described in the SDK documentation as below.

If device implementations report android.software.home_screen , they:

- [C-1-1] MUST honor the android.settings.HOME_SETTINGS intent to show a default app settings menu for Home Screen.

If device implementations report android.hardware.telephony , they:

- [C-2-1] MUST provide a settings menu that will call the android.provider.Telephony.ACTION_CHANGE_DEFAULT intent to show a dialog to change the default SMS application.
- [C-2-2] MUST honor the android.telecom.action.CHANGE_DEFAULT_DIALER intent to show a dialog to allow the user to change the default Phone application.
  - MUST use the user-selected default Phone app's UI for incoming and outgoing calls except for emergency calls, which would use the preinstalled Phone app.
- [C-2-3] MUST honor the android.telecom.action.CHANGE_PHONE_ACCOUNTS intent to provide user affordance to configure the ConnectionServices associated with the PhoneAccounts , as well as a default PhoneAccount that the telecommunications service provider will use to place outgoing calls. The AOSP implementation meets this requirement by including a "Calling Accounts option" menu within the "Calls" settings menu.
- [C-2-4] MUST allow android.telecom.CallRedirectionService for an app that holds the android.app.role.CALL_REDIRECTION role.
- [C-2-5] MUST provide the user affordance to choose an app that holds the android.app.role.CALL_REDIRECTION role.
- [C-2-6] MUST honor the android.intent.action.SENDTO and android.intent.action.VIEW intents and provide an activity to send/display SMS messages.
- [C-SR] Are Strongly Recommended to honor android.intent.action.ANSWER , android.intent.action.CALL , android.intent.action.CALL_BUTTON , android.intent.action.VIEW & android.intent.action.DIAL intents with a preloaded dialer application which can handle these intents and provide fulfillment as described in the SDK.

If device implementations report android.hardware.nfc.hce , they:

- [C-3-1] MUST honor the android.settings.NFC_PAYMENT_SETTINGS intent to show a default app settings menu for Tap and Pay.
- [C-3-2] MUST honor android.nfc.cardemulation.action.ACTION_CHANGE_DEFAULT intent to show an activity which opens a dialog to ask the user to change the default card emulation service for a certain category as described in the SDK.

If device implementations report android.hardware.nfc , they:

- [C-4-1] MUST honor these intents android.nfc.action.NDEF_DISCOVERED , android.nfc.action.TAG_DISCOVERED & android.nfc.action.TECH_DISCOVERED , to show an activity which fulfills developer expectations for these intents as described in the SDK.

If device implementations support the VoiceInteractionService and have more than one application using this API installed at a time, they:

- [C-4-1] MUST honor the android.settings.ACTION_VOICE_INPUT_SETTINGS intent to show a default app settings menu for voice input and assist.

If device implementations report android.hardware.bluetooth , they:

- [C-5-1] MUST honor the 'android.bluetooth.adapter.action.REQUEST_ENABLE' intent and show a system activity to allow the user to turn on Bluetooth.
- [C-5-2] MUST honor the 'android.bluetooth.adapter.action.REQUEST_DISCOVERABLE' intent and show a system activity that requests discoverable mode.

If device implementations support the DND feature, they:

- [C-6-1] MUST implement an activity that would respond to the intent ACTION_NOTIFICATION_POLICY_ACCESS_SETTINGS , which for implementations with UI_MODE_TYPE_NORMAL it MUST be an activity where the user can grant or deny the app access to DND policy configurations.

If device implementations allow users to use third-party input methods on the device, they:

- [C-7-1] MUST provide a user-accessible mechanism to add and configure third-party input methods in response to the android.settings.INPUT_METHOD_SETTINGS intent.

If device implementations support third-party accessibility services, they:

- [C-8-1] MUST honor the android.settings.ACCESSIBILITY_SETTINGS intent to provide a user-accessible mechanism to enable and disable the third-party accessibility services alongside the preloaded accessibility services.

If device implementations include support for Wi-Fi Easy Connect and expose the functionality to third-party apps, they:

- [C-9-1] MUST implement the Settings#ACTION_PROCESS_WIFI_EASY_CONNECT_URI Intent APIs as described in the SDK documentation.

If device implementations provide the data saver mode, they: * [C-10-1] MUST provide a user interface in the settings, that handles the Settings.ACTION_IGNORE_BACKGROUND_DATA_RESTRICTIONS_SETTINGS intent, allowing users to add applications to or remove applications from the allow list.

If device implementations do not provide the data saver mode, they:

- [C-11-1] MUST have an activity that handles the Settings.ACTION_IGNORE_BACKGROUND_DATA_RESTRICTIONS_SETTINGS intent but MAY implement it as a no-op.

If device implementations declare the support for camera via android.hardware.camera.any they:

- [C-12-1] MUST honor the android.media.action.STILL_IMAGE_CAMERA and android.media.action.STILL_IMAGE_CAMERA_SECURE intent and launch the camera in still image mode as described in the SDK.
- [C-12-2] MUST honor the android.media.action.VIDEO_CAMERA intent to launch the camera in video mode as described in the SDK.
- [C-12-3] MUST honor and only allow preinstalled Android applications to handle the following intents MediaStore.ACTION_IMAGE_CAPTURE , MediaStore.ACTION_IMAGE_CAPTURE_SECURE , and MediaStore.ACTION_VIDEO_CAPTURE as described in the SDK document .

If device implementations report android.software.device_admin , they:

- [C-13-1] MUST honor the intent android.app.action.ADD_DEVICE_ADMIN to invoke a UI to bring the user through adding the device administrator to the system (or allowing them to reject it).
- [C-13-2] MUST honor the intents android.app.action.ADMIN_POLICY_COMPLIANCE , android.app.action.GET_PROVISIONING_MODE , android.app.action.PROVISIONING_SUCCESSFUL , android.app.action.PROVISION_MANAGED_DEVICE , android.app.action.PROVISION_MANAGED_PROFILE , android.app.action.SET_NEW_PARENT_PROFILE_PASSWORD , android.app.action.SET_NEW_PASSWORD & android.app.action.START_ENCRYPTION and have an activity to provide fulfillment for these intents as described in SDK here .

If device implementations declare the android.software.autofill feature flag, they:

- [C-14-1] MUST fully implement the AutofillService and AutofillManager APIs and honor the android.settings.REQUEST_SET_AUTOFILL_SERVICE intent to show a default app settings menu to enable and disable autofill and change the default autofill service for the user.

If device implementations include a pre-installed app or wish to allow third-party apps to access the usage statistics, they:

- [C-SR] are STRONGLY RECOMMENDED provide user-accessible mechanism to grant or revoke access to the usage stats in response to the android.settings.ACTION_USAGE_ACCESS_SETTINGS intent for apps that declare the android.permission.PACKAGE_USAGE_STATS permission.

If device implementations intend to disallow any apps, including pre-installed apps, from accessing the usage statistics, they:

- [C-15-1] MUST still have an activity that handles the

android.settings.ACTION_USAGE_ACCESS_SETTINGS intent pattern but MUST implement it as a no-op, that is to have an equivalent behavior as when the user is declined for access.

If device implementations report the feature android.hardware.audio.output , they:

- [C-SR] Are Strongly Recommended to honor android.intent.action.TTS_SERVICE, android.speech.tts.engine.INSTALL_TTS_DATA & android.speech.tts.engine.GET_SAMPLE_TEXT intents have an activity to provide fulfillment for these intents as described in SDK here .

Android includes support for interactive screensavers, previously referred to as Dreams. Screen Savers allow users to interact with applications when a device connected to a power source is idle or docked in a desk dock. Device Implementations:

- SHOULD include support for screen savers and provide a settings option for users to configure screen savers in response to the android.settings.DREAM_SETTINGS intent.

3.2.4. Activities on secondary/multiple displays

If device implementations allow launching normal Android Activities on more than one display, they:

- [C-1-1] MUST set the android.software.activities_on_secondary_displays feature flag.
- [C-1-2] MUST guarantee API compatibility similar to an activity running on the primary display.
- [C-1-3] MUST land the new activity on the same display as the activity that launched it, when the new activity is launched without specifying a target display via the ActivityOptions.setLaunchDisplayId() API.
- [C-1-4] MUST destroy all activities, when a display with the Display.FLAG_PRIVATE flag is removed.
- [C-1-5] MUST securely hide content on all screens when the device is locked with a secure lock screen, unless the app opts in to show on top of lock screen using Activity#setShowWhenLocked() API.
- SHOULD have android.content.res.Configuration which corresponds to that display in order to be displayed, operate correctly, and maintain compatibility if an activity is launched on secondary display.

If device implementations allow launching normal Android Activities on secondary displays and a secondary display has the android.view.Display.FLAG_PRIVATE flag:

- [C-3-1] Only the owner of that display, system, and activities that are already on that display MUST be able to launch to it. Everyone can launch to a display that has android.view.Display.FLAG_PUBLIC flag.

3.3. Native API Compatibility

Native code compatibility is challenging. For this reason, device implementers are:

- [SR] STRONGLY RECOMMENDED to use the implementations of the libraries listed below from the upstream Android Open Source Project.

3.3.1. Application Binary Interfaces

Managed Dalvik bytecode can call into native code provided in the application .apk file as an ELF .so file compiled for the appropriate device hardware architecture. As native code is highly dependent on the underlying processor technology, Android defines a number of Application Binary Interfaces (ABIs) in the Android NDK.

Device implementations:

- [C-0-1] MUST be compatible with one or more defined ABIs and implement compatibility with the Android NDK.
- [C-0-2] MUST include support for code running in the managed environment to call into native code, using the standard Java Native Interface (JNI) semantics.
- [C-0-3] MUST be source-compatible (i.e. header-compatible) and binary-compatible (for the ABI) with each required library in the list below.

- [C-0-5] MUST accurately report the native Application Binary Interface (ABI) supported by the device, via the android.os.Build.SUPPORTED_ABIS , android.os.Build.SUPPORTED_32_BIT_ABIS , and android.os.Build.SUPPORTED_64_BIT_ABIS parameters, each a comma separated list of ABIs ordered from the most to the least preferred one.
- [C-0-6] MUST report, via the above parameters, a subset of the following list of ABIs and MUST NOT report any ABI not on the list.
  - armeabi
  - armeabi-v7a
  - arm64-v8a
  - x86
  - x86-64
  - [C-0-7] MUST make all the following libraries, providing native APIs, available to apps that include native code:
  - libaaudio.so (AAudio native audio support)
  - libamidi.so (native MIDI support, if feature android.software.midi is claimed as described in Section 5.9)
  - libandroid.so (native Android activity support)
  - libc (C library)
  - libcamera2ndk.so
  - libdl (dynamic linker)
  - libEGL.so (native OpenGL surface management)
  - libGLESv1_CM.so (OpenGL ES 1.x)
  - libGLESv2.so (OpenGL ES 2.0)
  - libGLESv3.so (OpenGL ES 3.x)
  - libicui18n.so
  - libicuuc.so
  - libjnigraphics.so
  - liblog (Android logging)
  - libmediandk.so (native media APIs support)
  - libm (math library)
  - libneuralnetworks.so (Neural Networks API)
  - libOpenMAXAL.so (OpenMAX AL 1.0.1 support)
  - libOpenSLES.so (OpenSL ES 1.0.1 audio support)
  - libRS.so
  - libstdc++ (Minimal support for C++)
  - libvulkan.so (Vulkan)
  - libz (Zlib compression)
  - JNI interface
- [C-0-8] MUST NOT add or remove the public functions for the native libraries listed above.
- [C-0-9] MUST list additional non-AOSP libraries exposed directly to third-party apps in /vendor/etc/public.libraries.txt .
- [C-0-10] MUST NOT expose any other native libraries, implemented and provided in AOSP as system libraries, to third-party apps targeting API level 24 or higher as they are reserved.
- [C-0-11] MUST export all the OpenGL ES 3.1 and Android Extension Pack function symbols, as defined in the NDK, through the libGLESv3.so library. Note that while all the symbols MUST be present, section 7.1.4.1 describes in more detail the requirements for when the full implementation of each corresponding functions are expected.
- [C-0-12] MUST export function symbols for the core Vulkan 1.0 function symbols, as well as the VK_KHR_surface , VK_KHR_android_surface , VK_KHR_swapchain , VK_KHR_maintenance1 , and VK_KHR_get_physical_device_properties2 extensions through the libvulkan.so library. Note that while all the symbols MUST be present, section 7.1.4.2 describes in more detail the requirements for when the full implementation of each corresponding functions are expected.
- SHOULD be built using the source code and header files available in the upstream Android Open Source Project

Note that future releases of Android may introduce support for additional ABIs.

3.3.2. 32-bit ARM Native Code Compatibility

If device implementations report the support of the armeabi ABI, they:

- [C-3-1] MUST also support armeabi-v7a and report its support, as armeabi is only for backwards compatibility with older apps.

If device implementations report the support of the armeabi-v7a ABI, for apps using this ABI, they:

- [C-2-1] MUST include the following lines in /proc/cpuinfo , and SHOULD NOT alter the values on the same device, even when they are read by other ABIs.
  - Features: , followed by a list of any optional ARMv7 CPU features supported by the device.
  - CPU architecture: , followed by an integer describing the device's highest supported ARM architecture (e.g., "8" for ARMv8 devices).
- [C-2-2] MUST always keep the following operations available, even in the case where the ABI is implemented on an ARMv8 architecture, either through native CPU support or through software emulation:
  - SWP and SWPB instructions.
  - SETEND instruction.
  - CP15ISB, CP15DSB, and CP15DMB barrier operations.
- [C-2-3] MUST include support for the Advanced SIMD (a.k.a. NEON) extension.

3.4. Web Compatibility

3.4.1. WebView Compatibility

If device implementations provide a complete implementation of the android.webkit.Webview API, they:

- [C-1-1] MUST report android.software.webview .
- [C-1-2] MUST use the Chromium Project build from the upstream Android Open Source Project on the Android 11 branch for the implementation of the android.webkit.WebView API.
- [C-1-3] The user agent string reported by the WebView MUST be in this format:
  Mozilla/5.0 (Linux; Android $(VERSION); [$(MODEL)] [Build/$(BUILD)]; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 $(CHROMIUM_VER) Mobile Safari/537.36
  - The value of the $(VERSION) string MUST be the same as the value for android.os.Build.VERSION.RELEASE.
  - The $(MODEL) string MAY be empty, but if it is not empty it MUST have the same value as android.os.Build.MODEL.
  - "Build/$(BUILD)" MAY be omitted, but if it is present the $(BUILD) string MUST be the same as the value for android.os.Build.ID.
  - The value of the $(CHROMIUM_VER) string MUST be the version of Chromium in the upstream Android Open Source Project.
  - Device implementations MAY omit Mobile in the user agent string.
- The WebView component SHOULD include support for as many HTML5 features as possible and if it supports the feature SHOULD conform to the HTML5 specification .
- [C-1-3] MUST render the provided content or remote URL content in a process that is distinct from the application that instantiates the WebView. Specifically the separate renderer process MUST hold lower privilege, run as a separate user ID, have no access to the app's data directory, have no direct network access, and only have access to the minimum-required system services over Binder. The AOSP implementation of WebView meets this requirement.

Note that if device implementations are 32-bit or declare the feature flag android.hardware.ram.low , they are exempted from C-1-3.

3.4.2. Browser Compatibility

If device implementations include a standalone Browser application for general web browsing, they:

- [C-1-1] MUST support each of these APIs associated with HTML5:

    - application cache/offline operation
    - <video> tag
    - geolocation
- [C-1-2] MUST support the HTML5/W3C webstorage API and SHOULD support the HTML5/W3C IndexedDB API . Note that as the web development standards bodies are transitioning to favor IndexedDB over webstorage, IndexedDB is expected to become a required component in a future version of Android.
- MAY ship a custom user agent string in the standalone Browser application.
- SHOULD implement support for as much of HTML5 as possible on the standalone Browser application (whether based on the upstream WebKit Browser application or a third-party replacement).

However, If device implementations do not include a standalone Browser application, they:

- [C-2-1] MUST still support the public intent patterns as described in section 3.2.3.1 .

3.5. API Behavioral Compatibility

Device implementations:

- [C-0-9] MUST ensure that API behavioral compatibility is applied for all installed apps unless they are restricted as described in Section 3.5.1 .
- [C-0-10] MUST NOT implement the whitelisting approach that ensures API behavioral compatibility only for apps that are selected by device implementers.

The behaviors of each of the API types (managed, soft, native, and web) must be consistent with the preferred implementation of the upstream Android Open Source Project . Some specific areas of compatibility are:

- [C-0-1] Devices MUST NOT change the behavior or semantics of a standard intent.
- [C-0-2] Devices MUST NOT alter the lifecycle or lifecycle semantics of a particular type of system component (such as Service, Activity, ContentProvider, etc.).
- [C-0-3] Devices MUST NOT change the semantics of a standard permission.
- Devices MUST NOT alter the limitations enforced on background applications. More specifically, for background apps:
    - [C-0-4] they MUST stop executing callbacks that are registered by the app to receive outputs from the GnssMeasurement and GnssNavigationMessage .
    - [C-0-5] they MUST rate-limit the frequency of updates that are provided to the app through the LocationManager API class or the WifiManager.startScan() method.
    - [C-0-6] if the app is targeting API level 25 or higher, they MUST NOT allow to register broadcast receivers for the implicit broadcasts of standard Android intents in the app's manifest, unless the broadcast intent requires a "signature" or "signatureOrSystem" protectionLevel permission or are on the exemption list .
    - [C-0-7] if the app is targeting API level 25 or higher, they MUST stop the app's background services, just as if the app had called the services' stopSelf() method, unless the app is placed on a temporary whitelist to handle a task that's visible to the user.
    - [C-0-8] if the app is targeting API level 25 or higher, they MUST release the wakelocks the app holds.
- [C-0-9] Devices MUST return the following security providers as the first seven array values from the Security.getProviders() method, in the given order and with the given names (as returned by Provider.getName()) and classes, unless the app has modified the list via insertProviderAt() or removeProvider() . Devices MAY return additional providers after the specified list of providers below.
    1. **AndroidNSSP** - android.security.net.config.NetworkSecurityConfigProvider
    2. **AndroidOpenSSL** - com.android.org.conscrypt.OpenSSLProvider
    3. **CertPathProvider** - sun.security.provider.CertPathProvider
    4. **AndroidKeyStoreBCWorkaround** - android.security.keystore.AndroidKeyStoreBCWorkaroundProvider
    5. **BC** - com.android.org.bouncycastle.jce.provider.BouncyCastleProvider
    6. **HarmonyJSSE** - com.android.org.conscrypt.JSSEProvider
    7. **AndroidKeyStore** - android.security.keystore.AndroidKeyStoreProvider

The above list is not comprehensive. The Compatibility Test Suite (CTS) tests significant portions of the platform for behavioral compatibility, but not all. It is the responsibility of the implementer to ensure behavioral compatibility with the Android Open Source Project. For this reason, device implementers SHOULD use the source code available via the Android Open Source Project where possible, rather than re-implement significant parts of the system.

3.5.1. Application Restriction

If device implementations implement a proprietary mechanism to restrict apps and that mechanism is more restrictive than the Rare App Standby Bucket , they:

- [C-1-1] MUST provide user affordance where the user can see the list of restricted apps.
- [C-1-2] MUST provide user affordance to turn on / off the restrictions on each app.
- [C-1-3] MUST not automatically apply restrictions without evidence of poor system health behavior, but MAY apply the restrictions on apps upon detection of poor system health behavior like stuck wakelocks, long running services, and other criteria. The criteria MAY be determined by device implementers but MUST be related to the app's impact on the system health. Other criteria that are not purely related to the system health, such as the app's lack of popularity in the market, MUST NOT be used as criteria.
- [C-1-4] MUST not automatically apply app restrictions for apps when a user has turned off app restrictions manually, and MAY suggest the user to apply app restrictions.
- [C-1-5] MUST inform users if app restrictions are applied to an app automatically. Such information MUST be provided within 24 hours of when the restrictions are applied.
- [C-1-6] MUST return `true` for ActivityManager.isBackgroundRestricted() when the restricted app calls this API.
- [C-1-7] MUST NOT restrict the top foreground app that is explicitly used by the user.
- [C-1-8] MUST suspend restrictions on an app that becomes the top foreground application when the user explicitly starts to use the app that used to be restricted.
- [C-1-9] MUST report all app restriction events via UsageStats . If device implementations extend the app restrictions that are implemented in AOSP, MUST follow the implementation described in this document .

3.6. API Namespaces

Android follows the package and class namespace conventions defined by the Java programming language. To ensure compatibility with third-party applications, device implementers MUST NOT make any prohibited modifications (see below) to these package namespaces:

- `java.*`
- `javax.*`
- `sun.*`
- `android.*`
- `androidx.*`
- `com.android.*`

That is, they:

- [C-0-1] MUST NOT modify the publicly exposed APIs on the Android platform by changing any method or class signatures, or by removing classes or class fields.
- [C-0-2] MUST NOT add any publicly exposed elements (such as classes or interfaces, or fields or methods to existing classes or interfaces) or Test or System APIs to the APIs in the above namespaces. A "publicly exposed element" is any construct that is not decorated with the "@hide" marker as used in the upstream Android source code.

Device implementers MAY modify the underlying implementation of the APIs, but such modifications:

- [C-0-3] MUST NOT impact the stated behavior and Java-language signature of any publicly exposed APIs.
- [C-0-4] MUST NOT be advertised or otherwise exposed to developers.

However, device implementers MAY add custom APIs outside the standard Android namespace, but the custom APIs:

- [C-0-5] MUST NOT be in a namespace owned by or referring to another organization. For instance, device implementers MUST NOT add APIs to the `com.google.*` or similar

namespace: only Google may do so. Similarly, Google MUST NOT add APIs to other companies' namespaces.

- [C-0-6] MUST be packaged in an Android shared library so that only apps that explicitly use them (via the <uses-library> mechanism) are affected by the increased memory usage of such APIs.

If a device implementer proposes to improve one of the package namespaces above (such as by adding useful new functionality to an existing API, or adding a new API), the implementer SHOULD visit source.android.com and begin the process for contributing changes and code, according to the information on that site.

Note that the restrictions above correspond to standard conventions for naming APIs in the Java programming language; this section simply aims to reinforce those conventions and make them binding through inclusion in this Compatibility Definition.

3.7. Runtime Compatibility

Device implementations:

- [C-0-1] MUST support the full Dalvik Executable (DEX) format and Dalvik bytecode specification and semantics .
- [C-0-2] MUST configure Dalvik runtimes to allocate memory in accordance with the upstream Android platform, and as specified by the following table. (See section 7.1.1 for screen size and screen density definitions.)
- SHOULD use Android RunTime (ART), the reference upstream implementation of the Dalvik Executable Format, and the reference implementation's package management system.
- SHOULD run fuzz tests under various modes of execution and target architectures to assure the stability of the runtime. Refer to JFuzz and DexFuzz in the Android Open Source Project website.

Note that memory values specified below are considered minimum values and device implementations MAY allocate more memory per application.

Screen Layout	Screen Density	Minimum Application Memory
Android Watch	120 dpi (ldpi)	32MB
	140 dpi (140dpi)	
	160 dpi (mdpi)	
	180 dpi (180dpi)	
	200 dpi (200dpi)	
	213 dpi (tvdpi)	
	220 dpi (220dpi)	36MB
	240 dpi (hdpi)	
	280 dpi (280dpi)	
	320 dpi (xhdpi)	48MB
	360 dpi (360dpi)	
	400 dpi (400dpi)	56MB
	420 dpi (420dpi)	64MB
	480 dpi (xxhdpi)	88MB
	560 dpi (560dpi)	112MB
	640 dpi (xxxhdpi)	154MB
	120 dpi (ldpi)	32MB
	140 dpi (140dpi)	
	160 dpi (mdpi)	
	180 dpi (180dpi)	

small/normal	200 dpi (200dpi) 213 dpi (tvdpi)	48MB
	220 dpi (220dpi)	
	240 dpi (hdpi)	
	280 dpi (280dpi)	
	320 dpi (xhdpi)	80MB
	360 dpi (360dpi)	
	400 dpi (400dpi)	96MB
	420 dpi (420dpi)	112MB
	480 dpi (xxhdpi)	128MB
	560 dpi (560dpi)	192MB
	640 dpi (xxxhdpi)	256MB
large	120 dpi (ldpi)	32MB
	140 dpi (140dpi)	48MB
	160 dpi (mdpi)	
	180 dpi (180dpi)	80MB
	200 dpi (200dpi)	
	213 dpi (tvdpi)	
	220 dpi (220dpi)	
	240 dpi (hdpi)	
	280 dpi (280dpi)	96MB
	320 dpi (xhdpi)	128MB
	360 dpi (360dpi)	160MB
	400 dpi (400dpi)	192MB
	420 dpi (420dpi)	228MB
	480 dpi (xxhdpi)	256MB
	560 dpi (560dpi)	384MB
	640 dpi (xxxhdpi)	512MB
xlarge	120 dpi (ldpi)	48MB
	140 dpi (140dpi)	80MB
	160 dpi (mdpi)	
	180 dpi (180dpi)	96MB
	200 dpi (200dpi)	
	213 dpi (tvdpi)	
	220 dpi (220dpi)	
	240 dpi (hdpi)	
	280 dpi (280dpi)	144MB
	320 dpi (xhdpi)	192MB
	360 dpi (360dpi)	240MB
	400 dpi (400dpi)	288MB
	420 dpi (420dpi)	336MB
	480 dpi (xxhdpi)	384MB
	560 dpi (560dpi)	576MB
	640 dpi (xxxhdpi)	768MB

3.8. User Interface Compatibility

3.8.1. Launcher (Home Screen)

Android includes a launcher application (home screen) and support for third-party applications to replace the device launcher (home screen).

If device implementations allow third-party applications to replace the device home screen, they:

- [C-1-1] MUST declare the platform feature android.software.home_screen .
- [C-1-2] MUST return the AdaptiveIconDrawable object when the third-party application use <adaptive-icon> tag to provide their icon, and the PackageManager methods to retrieve icons are called.

If device implementations include a default launcher that supports in-app pinning of shortcuts, they:

- [C-2-1] MUST report true for ShortcutManager.isRequestPinShortcutSupported() .
- [C-2-2] MUST have user affordance asking the user before adding a shortcut requested by apps via the ShortcutManager.requestPinShortcut() API method.
- [C-2-3] MUST support pinned shortcuts and dynamic and static shortcuts as documented on the App Shortcuts page .

Conversely, if device implementations do not support in-app pinning of shortcuts, they:

- [C-3-1] MUST report false for ShortcutManager.isRequestPinShortcutSupported() .

If device implementations implement a default launcher that provides quick access to the additional shortcuts provided by third-party apps through the ShortcutManager API, they:

- [C-4-1] MUST support all documented shortcut features (e.g. static and dynamic shortcuts, pinning shortcuts) and fully implement the APIs of the ShortcutManager API class.

If device implementations include a default launcher app that shows badges for the app icons, they:

- [C-5-1] MUST respect the NotificationChannel.setShowBadge() API method. In other words, show a visual affordance associated with the app icon if the value is set as true , and do not show any app icon badging scheme when all of the app's notification channels have set the value as false .
- MAY override the app icon badges with their proprietary badging scheme when third-party applications indicate support of the proprietary badging scheme through the use of proprietary APIs, but SHOULD use the resources and values provided through the notification badges APIs described in the SDK , such as the Notification.Builder.setNumber() and the Notification.Builder.setBadgeIconType() API.

3.8.2. Widgets

Android supports third-party app widgets by defining a component type and corresponding API and lifecycle that allows applications to expose an "AppWidget" to the end user.

If device implementations support third-party app widgets, they:

- [C-1-1] MUST declare support for platform feature android.software.app_widgets .
- [C-1-2] MUST include built-in support for AppWidgets and expose user interface affordances to add, configure, view, and remove AppWidgets directly within the Launcher.
- [C-1-3] MUST be capable of rendering widgets that are 4 x 4 in the standard grid size. See the App Widget DesignGuidelines in the Android SDK documentation for details.
- MAY support application widgets on the lock screen.

If device implementations support third-party app widgets and in-app pinning of shortcuts, they:

- [C-2-1] MUST report true for AppWidgetManager.html.isRequestPinAppWidgetSupported() .
- [C-2-2] MUST have user affordance asking the user before adding a shortcut requested by apps via the AppWidgetManager.requestPinAppWidget() API method.

3.8.3. Notifications

Android includes Notification and NotificationManager APIs that allow third-party app developers to notify users of notable events and attract users' attention using the hardware components (e.g.

sound, vibration and light) and software features (e.g. notification shade, system bar) of the device.

3.8.3.1. Presentation of Notifications

If device implementations allow third-party apps to notify users of notable events , they:

- [C-1-1] MUST support notifications that use hardware features, as described in the SDK documentation, and to the extent possible with the device implementation hardware. For instance, if a device implementation includes a vibrator, it MUST correctly implement the vibration APIs. If a device implementation lacks hardware, the corresponding APIs MUST be implemented as no-ops. This behavior is further detailed in section 7 .
- [C-1-2] MUST correctly render all resources (icons, animation files, etc.) provided for in the APIs, or in the Status/System Bar icon style guide , although they MAY provide an alternative user experience for notifications than that provided by the reference Android Open Source implementation.
- [C-1-3] MUST honor and implement properly the behaviors described for the APIs to update, remove and group notifications.
- [C-1-4] MUST provide the full behavior of the NotificationChannel API documented in the SDK.
- [C-1-5] MUST provide a user affordance to block and modify a certain third-party app's notification per each channel and app package level.
- [C-1-6] MUST also provide a user affordance to display deleted notification channels.
- [C-1-7] MUST correctly render all resources (images, stickers, icons, etc.) provided through Notification.MessagingStyle alongside the notification text without additional user interaction. For example, MUST show all resources including icons provided through android.app.Person in a group conversation that is set through setGroupConversation .
- [C-SR] Are STRONGLY RECOMMENDED to automatically surface a user affordance to block a certain third-party app's notification per each channel and app package level after the user dismisses that notification multiple times.
- SHOULD support rich notifications.
- SHOULD present some higher priority notifications as heads-up notifications.
- SHOULD have a user affordance to snooze notifications.
- MAY only manage the visibility and timing of when third-party apps can notify users of notable events to mitigate safety issues such as driver distraction.

Android 11 introduces support for conversation notifications, which are notifications that use MessagingStyle and provides a published People Shortcut ID.

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to group and display conversation notifications ahead of non conversation notifications with the exception of ongoing foreground service notifications and importance:high notifications.

If device implementations support conversation notifications and the app provides the required data for bubbles , they:

- [C-SR] Are STRONGLY RECOMMENDED to display this conversation as a bubble. The AOSP implementation meets these requirements with the default System UI, Settings, and Launcher.

If device implementations support rich notifications, they:

- [C-2-1] MUST use the exact resources as provided through the Notification.Style API class and its subclasses for the presented resource elements.
- SHOULD present each and every resource element (e.g. icon, title and summary text) defined in the Notification.Style API class and its subclasses.

If device implementations support heads-up notifications: they:

- [C-3-1] MUST use the heads-up notification view and resources as described in the Notification.Builder API class when heads-up notifications are presented.
- [C-3-2] MUST display the actions provided through Notification.Builder.addAction() together with the notification content without additional user interaction as described in the SDK .

3.8.3.2. Notification Listener Service

Android includes the NotificationListenerService APIs that allow apps (once explicitly enabled by the user) to receive a copy of all notifications as they are posted or updated.

Device implementations:

- [C-0-1] MUST correctly and promptly update notifications in their entirety to all such installed and user-enabled listener services, including any and all metadata attached to the Notification object.
- [C-0-2] MUST respect the snoozeNotification() API call, and dismiss the notification and make a callback after the snooze duration that is set in the API call.

If device implementations have a user affordance to snooze notifications, they:

- [C-1-1] MUST reflect the snoozed notification status properly through the standard APIs such as NotificationListenerService.getSnoozedNotifications() .
- [C-1-2] MUST make this user affordance available to snooze notifications from each installed third-party app's, unless they are from persistent/foreground services.

3.8.3.3. DND (Do not Disturb)

If device implementations support the DND feature, they:

- [C-1-1] MUST, for when the device implementation has provided a means for the user to grant or deny third-party apps to access the DND policy configuration, display Automatic DND rules created by applications alongside the user-created and pre-defined rules.
- [C-1-3] MUST honor the suppressedVisualEffects values passed along the NotificationManager.Policy and if an app has set any of the SUPPRESSED_EFFECT_SCREEN_OFF or SUPPRESSED_EFFECT_SCREEN_ON flags, it SHOULD indicate to the user that the visual effects are suppressed in the DND settings menu.

3.8.4. Search

Android includes APIs that allow developers to incorporate search into their applications and expose their application's data into the global system search. Generally speaking, this functionality consists of a single, system-wide user interface that allows users to enter queries, displays suggestions as users type, and displays results. The Android APIs allow developers to reuse this interface to provide search within their own apps and allow developers to supply results to the common global search user interface.

- Android device implementations SHOULD include global search, a single, shared, system-wide search user interface capable of real-time suggestions in response to user input.

If device implementations implement the global search interface, they:

- [C-1-1] MUST implement the APIs that allow third-party applications to add suggestions to the search box when it is run in global search mode.

If no third-party applications are installed that make use of the global search:

- The default behavior SHOULD be to display web search engine results and suggestions.

Android also includes the Assist APIs to allow applications to elect how much information of the current context is shared with the assistant on the device.

If device implementations support the Assist action, they:

- [C-2-1] MUST indicate clearly to the end user when the context is shared, by either:
  - Each time the assist app accesses the context, displaying a white light around the edges of the screen that meet or exceed the duration and brightness of the Android Open Source Project implementation.
  - For the preinstalled assist app, providing a user affordance less than two navigations away from the default voice input and assistant app settings menu , and only sharing the context when the assist app is explicitly invoked by the user through a hotword or assist navigation key input.
- [C-2-2] The designated interaction to launch the assist app as described in section 7.2.3 MUST launch the user-selected assist app, in other words the app that implements

VoiceInteractionService , or an activity handling the ACTION_ASSIST intent.

3.8.5. Alerts and Toasts

Applications can use the Toast API to display short non-modal strings to the end user that disappear after a brief period of time, and use the TYPE_APPLICATION_OVERLAY window type API to display alert windows as an overlay over other apps.

If device implementations include a screen or video output, they:

- [C-1-1] MUST provide a user affordance to block an app from displaying alert windows that use the TYPE_APPLICATION_OVERLAY . The AOSP implementation meets this requirement by having controls in the notification shade.
- [C-1-2] MUST honor the Toast API and display Toasts from applications to end users in some highly visible manner.

3.8.6. Themes

Android provides “themes” as a mechanism for applications to apply styles across an entire Activity or application.

Android includes a “Holo” and "Material" theme family as a set of defined styles for application developers to use if they want to match the Holo theme look and feel as defined by the Android SDK.

If device implementations include a screen or video output, they:

- [C-1-1] MUST NOT alter any of the Holo theme attributes exposed to applications.
- [C-1-2] MUST support the “Material” theme family and MUST NOT alter any of the Material theme attributes or their assets exposed to applications.
- [C-1-3] MUST use the Roboto version 2.x fonts as the out-of-box Sans-serif font family for languages that Roboto supports.

Android also includes a “Device Default” theme family as a set of defined styles for application developers to use if they want to match the look and feel of the device theme as defined by the device implementer.

- Device implementations MAY modify the Device Default theme attributes exposed to applications.

Android supports a variant theme with translucent system bars, which allows application developers to fill the area behind the status and navigation bar with their app content. To enable a consistent developer experience in this configuration, it is important the status bar icon style is maintained across different device implementations.

If device implementations include a system status bar, they:

- [C-2-1] MUST use white for system status icons (such as signal strength and battery level) and notifications issued by the system, unless the icon is indicating a problematic status or an app requests a light status bar using the SYSTEM_UI_FLAG_LIGHT_STATUS_BAR flag.
- [C-2-2] Android device implementations MUST change the color of the system status icons to black (for details, refer to R.style) when an app requests a light status bar.

3.8.7. Live Wallpapers

Android defines a component type and corresponding API and lifecycle that allows applications to expose one or more “Live Wallpapers” to the end user. Live wallpapers are animations, patterns, or similar images with limited input capabilities that display as a wallpaper, behind other applications.

Hardware is considered capable of reliably running live wallpapers if it can run all live wallpapers, with no limitations on functionality, at a reasonable frame rate with no adverse effects on other applications. If limitations in the hardware cause wallpapers and/or applications to crash, malfunction, consume excessive CPU or battery power, or run at unacceptably low frame rates, the hardware is considered incapable of running live wallpaper. As an example, some live wallpapers may use an OpenGL 2.0 or 3.x context to render their content. Live wallpaper will not run reliably on hardware that does not support multiple OpenGL contexts because the live wallpaper use of an OpenGL context may conflict with other applications that also use an OpenGL context.

- Device implementations capable of running live wallpapers reliably as described above SHOULD implement live wallpapers.

If device implementations implement live wallpapers, they:

- [C-1-1] MUST report the platform feature flag android.software.live_wallpaper.

3.8.8. Activity Switching

The upstream Android source code includes the overview screen , a system-level user interface for task switching and displaying recently accessed activities and tasks using a thumbnail image of the application's graphical state at the moment the user last left the application.

Device implementations including the recents function navigation key as detailed in section 7.2.3 MAY alter the interface.

If device implementations including the recents function navigation key as detailed in section 7.2.3 alter the interface, they:

- [C-1-1] MUST support at least up to 7 displayed activities.
- SHOULD at least display the title of 4 activities at a time.
- [C-1-2] MUST implement the screen pinning behavior and provide the user with a settings menu to toggle the feature.
- SHOULD display highlight color, icon, screen title in recents.
- SHOULD display a closing affordance ("x") but MAY delay this until user interacts with screens.
- SHOULD implement a shortcut to switch easily to the previous activity.
- SHOULD trigger the fast-switch action between the two most recently used apps, when the recents function key is tapped twice.
- SHOULD trigger the split-screen multiwindow-mode, if supported, when the recents functions key is long pressed.
- MAY display affiliated recents as a group that moves together.
- [SR] Are STRONGLY RECOMMENDED to use the upstream Android user interface (or a similar thumbnail-based interface) for the overview screen.

3.8.9. Input Management

Android includes support for Input Management and support for third-party input method editors.

If device implementations allow users to use third-party input methods on the device, they:

- [C-1-1] MUST declare the platform feature android.software.input_methods and support IME APIs as defined in the Android SDK documentation.

3.8.10. Lock Screen Media Control

The Remote Control Client API is deprecated from Android 5.0 in favor of the Media Notification Template that allows media applications to integrate with playback controls that are displayed on the lock screen.

3.8.11. Screen savers (previously Dreams)

See section 3.2.3.5 for settings intent to congfigure screen savers.

3.8.12. Location

If device implementations include a hardware sensor (e.g. GPS) that is capable of providing the location coordinates, they

- [C-1-2] MUST display the current status of location in the Location menu within Settings.
- [C-1-3] MUST NOT display location modes in the Location menu within Settings.

3.8.13. Unicode and Font

Android includes support for the emoji characters defined in Unicode 10.0 .

If device implementations include a screen or video output, they:

- [C-1-1] MUST be capable of rendering these emoji characters in color glyph.
- [C-1-2] MUST include support for:

    - Roboto 2 font with different weights—sans-serif-thin, sans-serif-light, sans-serif-medium, sans-serif-black, sans-serif-condensed, sans-serif-condensed-light for the languages available on the device.
    - Full Unicode 7.0 coverage of Latin, Greek, and Cyrillic, including the Latin Extended A, B, C, and D ranges, and all glyphs in the currency symbols block of Unicode 7.0.
- SHOULD support the skin tone and diverse family emojis as specified in the Unicode Technical Report #51 .

If device implementations include an IME, they:

- SHOULD provide an input method to the user for these emoji characters.

Android includes support to render Myanmar fonts. Myanmar has several non-Unicode compliant fonts, commonly known as "Zawgyi," for rendering Myanmar languages.
If device implementations include support for Burmese, they:

* [C-2-1] MUST render text with Unicode compliant font as default; non-Unicode compliant font MUST NOT be set as default font unless the user chooses it in the language picker.
* [C-2-2] MUST support a Unicode font and a non-Unicode compliant font if a non-Unicode compliant font is supported on the device. Non-Unicode compliant font MUST NOT remove or overwrite the Unicode font.
* [C-2-3] MUST render text with non-Unicode compliant font ONLY IF a language code with [script code Qaag](http://unicode.org/reports/tr35/#unicode_script_subtag_validity) is specified (e.g. my-Qaag). No other ISO language or region codes (whether assigned, unassigned, or reserved) can be used to refer to non-Unicode compliant font for Myanmar. App developers and web page authors can specify my-Qaag as the designated language code as they would for any other language.

3.8.14. Multi-windows

If device implementations have the capability to display multiple activities at the same time, they:

- [C-1-1] MUST implement such multi-window mode(s) in accordance with the application behaviors and APIs described in the Android SDK multi-window mode support documentation and meet the following requirements:
- [C-1-2] MUST honor android:resizeableActivity that is set by an app in the AndroidManifest.xml file as described in this SDK .
- [C-1-3] MUST NOT offer split-screen or freeform mode if the screen height is less than 440 dp and the screen width is less than 440 dp.
- [C-1-4] An activity MUST NOT be resized to a size smaller than 220dp in multi-window modes other than Picture-in-Picture.
- Device implementations with screen size xlarge SHOULD support freeform mode.

If device implementations support multi-window mode(s), and the split screen mode, they:

- [C-2-1] MUST preload a resizeable launcher as the default.
- [C-2-2] MUST crop the docked activity of a split-screen multi-window but SHOULD show some content of it, if the Launcher app is the focused window.
- [C-2-3] MUST honor the declared AndroidManifestLayout_minWidth and AndroidManifestLayout_minHeight values of the third-party launcher application and not override these values in the course of showing some content of the docked activity.

If device implementations support multi-window mode(s) and Picture-in-Picture multi-window mode, they:

- [C-3-1] MUST launch activities in picture-in-picture multi-window mode when the app is: * Targeting API level 26 or higher and declares android:supportsPictureInPicture * Targeting API level 25 or lower and declares both android:resizeableActivity and android:supportsPictureInPicture .
- [C-3-2] MUST expose the actions in their SystemUI as specified by the current PIP activity through the setActions() API.
- [C-3-3] MUST support aspect ratios greater than or equal to 1:2.39 and less than or equal

to 2.39:1, as specified by the PIP activity through the setAspectRatio() API.

- [C-3-4] MUST use KeyEvent.KEYCODE_WINDOW to control the PIP window; if PIP mode is not implemented, the key MUST be available to the foreground activity.
- [C-3-5] MUST provide a user affordance to block an app from displaying in PIP mode; the AOSP implementation meets this requirement by having controls in the notification shade.
- [C-3-6] MUST allocate the following minimum width and height for the PIP window when an application does not declare any value for AndroidManifestLayout_minWidth and AndroidManifestLayout_minHeight :
  - Devices with the Configuration.uiMode that is set other than UI_MODE_TYPE_TELEVISION MUST allocate a minimum width and height of 108 dp.
  - Devices with the Configuration.uiMode that is set to UI_MODE_TYPE_TELEVISION MUST allocate a minimum width of 240 dp and a minimum height of 135 dp.

3.8.15. Display Cutout

Android supports a Display Cutout as described in the SDK document. The DisplayCutout API defines an area on the edge of the display that may not be functional for an application due to a display cutout or curved display on the edge(s).

If device implementations include display cutout(s), they:

- [C-1-5] MUST NOT have cutout(s) if the device's aspect ratio is 1.0(1:1).
- [C-1-2] MUST NOT have more than one cutout per edge.
- [C-1-3] MUST honor the display cutout flags set by the app through the WindowManager.LayoutParams API as described in the SDK.
- [C-1-4] MUST report correct values for all cutout metrics defined in the DisplayCutout API.

3.8.16. Device Controls

Android includes ControlsProviderService and Control APIs to allow third-party applications to publish device controls for quick status and action for users.

See Section 2_2_3 for device-specific requirements.

3.9. Device Administration

Android includes features that allow security-aware applications to perform device administration functions at the system level, such as enforcing password policies or performing remote wipe, through the Android Device Administration API .

If device implementations implement the full range of device administration policies defined in the Android SDK documentation, they:

- [C-1-1] MUST declare android.software.device_admin .
- [C-1-2] MUST support device owner provisioning as described in section 3.9.1 and section 3.9.1.1 .

3.9.1 Device Provisioning

3.9.1.1 Device owner provisioning

If device implementations declare android.software.device_admin , they:

- [C-1-1] MUST support enrolling a Device Policy Client (DPC) as aDevice Owner app as described below:
  - When the device implementation has no user data is configured yet, it:
    - [C-1-3] MUST report true for DevicePolicyManager.isProvisioningAllowed(ACTION_PROVISION_MANAGED_DEVICE) .
    - [C-1-4] MUST enroll the DPC application as the Device Owner app in response to the intent action android.app.action.PROVISION_MANAGED_DEVICE .
    - [C-1-5] MUST enroll the DPC application as the Device Owner app if

the device declares Near-Field Communications (NFC) support via the feature flag android.hardware.nfc and receives an NFC message containing a record with MIME type MIME_TYPE_PROVISIONING_NFC .

    - When the device implementation has user data, it:
        - [C-1-6] MUST report false for the DevicePolicyManager.isProvisioningAllowed(ACTION_PROVISION_MANAGED_DEVICE) .
        - [C-1-7] MUST not enroll any DPC application as the Device Owner App any more.
- [C-1-2] MUST require some affirmative action before or during the provisioning process to consent to the app being set as Device Owner. Consent can be via user action or by some programmatic means but appropriate disclosure notice (as referenced in AOSP) MUST be shown before device owner provisioning is initiated. Also, the programmatic device owner consent mechanism used (by enterprises) for device owner provisioning MUST NOT interfere with the Out-Of-Box Experience for non-enterprise use.
- [C-1-3] MUST NOT hard code the consent or prevent the use of other device owner apps.

If device implementations declare android.software.device_admin , but also include a proprietary Device Owner management solution and provide a mechanism to promote an application configured in their solution as a "Device Owner equivalent" to the standard "Device Owner" as recognized by the standard Android DevicePolicyManager APIs, they:

- [C-2-1] MUST have a process in place to verify that the specific app being promoted belongs to a legitimate enterprise device management solution and it has been already configured in the proprietary solution to have the rights equivalent as a "Device Owner".
- [C-2-2] MUST show the same AOSP Device Owner consent disclosure as the flow initiated by android.app.action.PROVISION_MANAGED_DEVICE prior to enrolling the DPC application as "Device Owner".
- MAY have user data on the device prior to enrolling the DPC application as "Device Owner".

3.9.1.2 Managed profile provisioning

If device implementations declare android.software.managed_users , they:

- [C-1-1] MUST implement the APIs allowing a Device Policy Controller (DPC) application to become the owner of a new Managed Profile .
- [C-1-2] The managed profile provisioning process (the flow initiated by android.app.action.PROVISION_MANAGED_PROFILE) users experience MUST align with the AOSP implementation.
- [C-1-3] MUST provide the following user affordances within the Settings to indicate to the user when a particular system function has been disabled by the Device Policy Controller (DPC):
    - A consistent icon or other user affordance (for example the upstream AOSP info icon) to represent when a particular setting is restricted by a Device Admin.
    - A short explanation message, as provided by the Device Admin via the setShortSupportMessage .
    - The DPC application's icon.

3.9.2 Managed Profile Support

If device implementations declare android.software.managed_users , they:

- [C-1-1] MUST support managed profiles via the android.app.admin.DevicePolicyManager APIs.
- [C-1-2] MUST allow one and only one managed profile to be created .
- [C-1-3] MUST use an icon badge (similar to the AOSP upstream work badge) to represent the managed applications and widgets and other badged UI elements like Recents & Notifications.
- [C-1-4] MUST display a notification icon (similar to the AOSP upstream work badge) to indicate when user is within a managed profile application.
- [C-1-5] MUST display a toast indicating that the user is in the managed profile if and when

the device wakes up (ACTION_USER_PRESENT) and the foreground application is within the managed profile.

- [C-1-6] Where a managed profile exists, MUST show a visual affordance in the Intent 'Chooser' to allow the user to forward the intent from the managed profile to the primary user or vice versa, if enabled by the Device Policy Controller.
- [C-1-7] Where a managed profile exists, MUST expose the following user affordances for both the primary user and the managed profile:
  - Separate accounting for battery, location, mobile data and storage usage for the primary user and managed profile.
  - Independent management of VPN Applications installed within the primary user or managed profile.
  - Independent management of applications installed within the primary user or managed profile.
  - Independent management of accounts within the primary user or managed profile.
- [C-1-8] MUST ensure the preinstalled dialer, contacts and messaging applications can search for and look up caller information from the managed profile (if one exists) alongside those from the primary profile, if the Device Policy Controller permits it.
- [C-1-9] MUST ensure that it satisfies all the security requirements applicable for a device with multiple users enabled (see section 9.5), even though the managed profile is not counted as another user in addition to the primary user.

If device implementations declare android.software.managed_users and android.software.secure_lock_screen , they:

- [C-2-1] MUST support the ability to specify a separate lock screen meeting the following requirements to grant access to apps running in a managed profile only.
  - Device implementations MUST honor the DevicePolicyManager.ACTION_SET_NEW_PASSWORD intent and show an interface to configure a separate lock screen credential for the managed profile.
  - The lock screen credentials of the managed profile MUST use the same credential storage and management mechanisms as the parent profile, as documented on the Android Open Source Project Site .
  - The DPC password policies MUST apply to only the managed profile's lock screen credentials unless called upon the DevicePolicyManager instance returned by getParentProfileInstance .
- When contacts from the managed profile are displayed in the preinstalled call log, in-call UI, in-progress and missed-call notifications, contacts and messaging apps they SHOULD be badged with the same badge used to indicate managed profile applications.

3.9.3 Managed User Support

If device implementations declare android.software.managed_users , they:

- [C-1-1] MUST provide a user affordance to logout from the current user and switch back to the primary user in multiple-user session when isLogoutEnabled returns true . The user affordance MUST be accessible from the lockscreen without unlocking the device.

3.10. Accessibility

Android provides an accessibility layer that helps users with disabilities to navigate their devices more easily. In addition, Android provides platform APIs that enable accessibility service implementations to receive callbacks for user and system events and generate alternate feedback mechanisms, such as text-to-speech, haptic feedback, and trackball/d-pad navigation.

If device implementations support third-party accessibility services, they:

- [C-1-1] MUST provide an implementation of the Android accessibility framework as described in the accessibility APIs SDK documentation.
- [C-1-2] MUST generate accessibility events and deliver the appropriate AccessibilityEvent to all registered AccessibilityService implementations as documented in the SDK.
- [C-1-4] MUST add a button in the system's navigation bar allowing the user to control the accessibility service when the enabled accessibility services declare the AccessibilityServiceInfo.FLAG_REQUEST_ACCESSIBILITY_BUTTON . Note that for device implementations with no system navigation bar, this requirement is not applicable, but

device implementations SHOULD provide a user affordance to control these accessibility services.

If device implementations include preinstalled accessibility services, they:

- [C-2-1] MUST implement these preinstalled accessibility services as Direct Boot Aware apps when the data storage is encrypted with File Based Encryption (FBE).
- SHOULD provide a mechanism in the out-of-box setup flow for users to enable relevant accessibility services, as well as options to adjust the font size, display size and magnification gestures.

3.11. Text-to-Speech

Android includes APIs that allow applications to make use of text-to-speech (TTS) services and allows service providers to provide implementations of TTS services.

If device implementations reporting the feature android.hardware.audio.output, they:

- [C-1-1] MUST support the Android TTS framework APIs.

If device implementations support installation of third-party TTS engines, they:

- [C-2-1] MUST provide user affordance to allow the user to select a TTS engine for use at system level.

3.12. TV Input Framework

The Android Television Input Framework (TIF) simplifies the delivery of live content to Android Television devices. TIF provides a standard API to create input modules that control Android Television devices.

If device implementations support TIF, they:

- [C-1-1] MUST declare the platform feature android.software.live_tv .
- [C-1-2] MUST support all TIF APIs such that an application which uses these APIs and the third-party TIF-based inputs service can be installed and used on the device.

3.13. Quick Settings

Android provides a Quick Settings UI component that allows quick access to frequently used or urgently needed actions.

If device implementations include a Quick Settings UI component, they:

- [C-1-1] MUST allow the user to add or remove the tiles provided through the quicksettings APIs from a third-party app.
- [C-1-2] MUST NOT automatically add a tile from a third-party app directly to the Quick Settings.
- [C-1-3] MUST display all the user-added tiles from third-party apps alongside the system-provided quick setting tiles.

3.14. Media UI

If device implementations include non-voice-activated applications (the Apps) that interact with third-party applications through MediaBrowser or MediaSession , the Apps:

- [C-1-2] MUST clearly display icons obtained via getIconBitmap() or getIconUri() and titles obtained via getTitle() as described in MediaDescription . May shorten titles to comply with safety regulations (e.g. driver distraction).
- [C-1-3] MUST show the third-party application icon whenever displaying content provided by this third-party application.
- [C-1-4] MUST allow the user to interact with the entire MediaBrowser hierarchy. MAY restrict the access to part of the hierarchy to comply with safety regulations (e.g. driver distraction), but MUST NOT give preferential treatment based on content or content provider.
- [C-1-5] MUST consider double tap of KEYCODE_HEADSETHOOK or

KEYCODE_MEDIA_PLAY_PAUSE as KEYCODE_MEDIA_NEXT for MediaSession.Callback#onMediaButtonEvent .

3.15. Instant Apps

Device implementations MUST satisfy the following requirements:

- [C-0-1] Instant Apps MUST only be granted permissions that have the android:protectionLevel set to "instant" .
- [C-0-2] Instant Apps MUST NOT interact with installed apps via implicit intents unless one of the following is true:
  - The component's intent pattern filter is exposed and has CATEGORY_BROWSABLE
  - The action is one of ACTION_SEND, ACTION_SENDTO, ACTION_SEND_MULTIPLE
  - The target is explicitly exposed with android:visibleToInstantApps
- [C-0-3] Instant Apps MUST NOT interact explicitly with installed apps unless the component is exposed via android:visibleToInstantApps.
- [C-0-4] Installed Apps MUST NOT see details about Instant Apps on the device unless the Instant App explicitly connects to the installed application.
- Device implementations MUST provide the following user affordances for interacting with Instant Apps. The AOSP meets the requirements with the default System UI, Settings, and Launcher. Device implementations:
  - [C-0-5] MUST provide a user affordance to view and delete Instant Apps locally cached for each individual app package.
  - [C-0-6] MUST provide a persistent user notification that can be collapsed while an Instant App is running in the foreground. This user notification MUST include that Instant Apps do not require installation and provide a user affordance that directs the user to the application info screen in Settings. For Instant Apps launched via web intents, as defined by using an intent with action set to Intent.ACTION_VIEW and with a scheme of "http" or "https", an additional user affordance SHOULD allow the user not to launch the Instant App and launch the associated link with the configured web browser, if a browser is available on the device.
  - [C-0-7] MUST allow running Instant Apps to be accessed from the Recents function if the Recents function is available on the device.

If device implementations support Instant Apps, they:

- [C-1-1] MUST preload one or more applications or service components with an intent handler for the intents listed in the SDK here and make the intents visible for Instant Apps.

3.16. Companion Device Pairing

Android includes support for companion device pairing to more effectively manage association with companion devices and provides the CompanionDeviceManager API for apps to access this feature.

If device implementations support the companion device pairing feature, they:

- [C-1-1] MUST declare the feature flag FEATURE_COMPANION_DEVICE_SETUP .
- [C-1-2] MUST ensure the APIs in the android.companion package is fully implemented.
- [C-1-3] MUST provide user affordances for the user to select/confirm a companion device is present and operational.

3.17. Heavyweight Apps

If device implementations declare the feature FEATURE_CANT_SAVE_STATE , then they:

- [C-1-1] MUST have only one installed app that specifies cantSaveState running in the system at a time. If the user leaves such an app without explicitly exiting it (for example by pressing home while leaving an active activity the system, instead of pressing back with no remaining active activities in the system), then device implementations MUST prioritize that app in RAM as they do for other things that are expected to remain running, such as foreground services. While such an app is in the background, the system can still apply power management features to it, such as limiting CPU and network access.

- [C-1-2] MUST provide a UI affordance to chose the app that won't participate in the normal state save/restore mechanism once the user launches a second app declared with cantSaveState attribute.
- [C-1-3] MUST NOT apply other changes in policy to apps that specify cantSaveState , such as changing CPU performance or changing scheduling prioritization.

If device implementations don't declare the feature FEATURE_CANT_SAVE_STATE , then they:

- [C-1-1] MUST ignore the cantSaveState attribute set by apps and MUST NOT change the app behavior based on that attribute.

3.18. Contacts

Android includes Contacts Provider APIs to allow applications to manage contact information stored on the device. Contact data that is entered directly into the device is typically synchronized with a web service, but the data MAY also only reside locally on the device. Contacts that are only stored on the device are referred to as **local contacts.**

RawContacts are "associated with" or "stored in" an Account when the ACCOUNT_NAME , and ACCOUNT_TYPE , columns for the raw contacts match the corresponding Account.name and Account.type fields of the account.

Default local account : an account for raw contacts that are only stored on the device and not associated with an Account in the AccountManager , which are created with *null* values for the ACCOUNT_NAME , and ACCOUNT_TYPE , columns.

Custom local account : an account for raw contacts that are only stored on the device and not associated with an Account in the AccountManager, which are created with *at least one non-null value* for the ACCOUNT_NAME , and ACCOUNT_TYPE , columns.

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to not create **custom local accounts** .

If device implementations use a **custom local account** :

- [C-1-1] The ACCOUNT_NAME , of the **custom local account** MUST be returned by ContactsContract.RawContacts.getLocalAccountName
- [C-1-2] The ACCOUNT_TYPE , of the **custom local account** MUST be returned by ContactsContract.RawContacts.getLocalAccountType
- [C-1-3] Raw contacts that are inserted by third party applications with the **default local account** (i.e. by setting null values for ACCOUNT_NAME and ACCOUNT_TYPE) MUST be inserted to the **custom local account** .
- [C-1-4] Raw contacts inserted into the **custom local account** MUST not be removed when accounts are added or removed.
- [C-1-5] Delete operations performed against the **custom local account** MUST result in raw contacts being purged immediately (as if the CALLER_IS_SYNCADAPTER param was set to true), even if the CALLER_IS_SYNCADAPTER param was set to false or not specified.

4. Application Packaging Compatibility

Devices implementations:

- [C-0-1] MUST be capable of installing and running Android ".apk" files as generated by the "aapt" tool included in the official Android SDK .
- As the above requirement may be challenging, device implementations are RECOMMENDED to use the AOSP reference implementation's package management system.

Device implementations:

- [C-0-2] MUST support verifying ".apk" files using the APK Signature Scheme v3 , APK Signature Scheme v2 and JAR signing .
- [C-0-3] MUST NOT extend either the .apk , Android Manifest , Dalvik bytecode , or RenderScript bytecode formats in such a way that would prevent those files from installing and running correctly on other compatible devices.
- [C-0-4] MUST NOT allow apps other than the current "installer of record" for the package to silently uninstall the app without any user confirmation, as documented in the SDK for

the DELETE_PACKAGE permission. The only exceptions are the system package verifier app handling PACKAGE_NEEDS_VERIFICATION intent and the storage manager app handling ACTION_MANAGE_STORAGE intent.

- [C-0-5] MUST have an activity that handles the android.settings.MANAGE_UNKNOWN_APP_SOURCES intent.
- [C-0-6] MUST NOT install application packages from unknown sources, unless the app that requests the installation meets all the following requirements:
  - It MUST declare the REQUEST_INSTALL_PACKAGES permission or have the android:targetSdkVersion set at 24 or lower.
  - It MUST have been granted permission by the user to install apps from unknown sources.
- SHOULD provide a user affordance to grant/revoke the permission to install apps from unknown sources per application, but MAY choose to implement this as a no-op and return RESULT_CANCELED for startActivityForResult() , if the device implementation does not want to allow users to have this choice. However, even in such cases, they SHOULD indicate to the user why there is no such choice presented.
- [C-0-7] MUST display a warning dialog with the warning string that is provided through the system API PackageManager.setHarmfulAppWarning to the user before launching an activity in an application that has been marked by the same system API PackageManager.setHarmfulAppWarning as potentially harmful.
- SHOULD provide a user affordance to choose to uninstall or launch an application on the warning dialog.
- [C-0-8] MUST implement support for Incremental File System as documented here .
- [C-0-9] MUST support verifying .apk files using the APK Signature Scheme v4 .
- If device implementations are already launched on an earlier Android version and cannot meet the requirements [C-0-8] and [C-0-9] through a system software update, they MAY be exempted from these requirements.

5. Multimedia Compatibility

Device implementations:

- [C-0-1] MUST support the media formats, encoders, decoders, file types, and container formats defined in section 5.1 for each and every codec declared by MediaCodecList .
- [C-0-2] MUST declare and report support of the encoders, decoders available to third-party applications via MediaCodecList .
- [C-0-3] MUST be able to properly decode and make available to third-party apps all the formats it can encode. This includes all bitstreams that its encoders generate and the profiles reported in its CamcorderProfile .

Device implementations:

- SHOULD aim for minimum codec latency, in others words, they
  - SHOULD NOT consume and store input buffers and return input buffers only once processed.
  - SHOULD NOT hold onto decoded buffers for longer than as specified by the standard (e.g. SPS).
  - SHOULD NOT hold onto encoded buffers longer than required by the GOP structure.

All of the codecs listed in the section below are provided as software implementations in the preferred Android implementation from the Android Open Source Project.

Please note that neither Google nor the Open Handset Alliance make any representation that these codecs are free from third-party patents. Those intending to use this source code in hardware or software products are advised that implementations of this code, including in open source software or shareware, may require patent licenses from the relevant patent holders.

5.1. Media Codecs

5.1.1. Audio Encoding

See more details in 5.1.3. Audio Codecs Details .

If device implementations declare android.hardware.microphone , they MUST support encoding the following audio formats and make them available to third-party apps:

- [C-1-1] PCM/WAVE
- [C-1-2] FLAC
- [C-1-3] Opus

All audio encoders MUST support:

- [C-3-1] PCM 16-bit native byte order audio frames via the android.media.MediaCodec API.

5.1.2. Audio Decoding

See more details in 5.1.3. Audio Codecs Details .

If device implementations declare support for the android.hardware.audio.output feature, they must support decoding the following audio formats:

- [C-1-1] MPEG-4 AAC Profile (AAC LC)
- [C-1-2] MPEG-4 HE AAC Profile (AAC+)
- [C-1-3] MPEG-4 HE AACv2 Profile (enhanced AAC+)
- [C-1-4] AAC ELD (enhanced low delay AAC)
- [C-1-11] xHE-AAC (ISO/IEC 23003-3 Extended HE AAC Profile, which includes the USAC Baseline Profile, and ISO/IEC 23003-4 Dynamic Range Control Profile)
- [C-1-5] FLAC
- [C-1-6] MP3
- [C-1-7] MIDI
- [C-1-8] Vorbis
- [C-1-9] PCM/WAVE including high-resolution audio formats up to 24 bits, 192 kHz sample rate, and 8 channels. Note that this requirement is for decoding only, and that a device is permitted to downsample and downmix during the playback phase.
- [C-1-10] Opus

If device implementations support the decoding of AAC input buffers of multichannel streams (i.e. more than two channels) to PCM through the default AAC audio decoder in the android.media.MediaCodec API, the following MUST be supported:

- [C-2-1] Decoding MUST be performed without downmixing (e.g. a 5.0 AAC stream must be decoded to five channels of PCM, a 5.1 AAC stream must be decoded to six channels of PCM).
- [C-2-2] Dynamic range metadata MUST be as defined in "Dynamic Range Control (DRC)" in ISO/IEC 14496-3, and the android.media.MediaFormat DRC keys to configure the dynamic range-related behaviors of the audio decoder. The AAC DRC keys were introduced in API 21, and are: KEY_AAC_DRC_ATTENUATION_FACTOR , KEY_AAC_DRC_BOOST_FACTOR , KEY_AAC_DRC_HEAVY_COMPRESSION , KEY_AAC_DRC_TARGET_REFERENCE_LEVEL and KEY_AAC_ENCODED_TARGET_LEVEL .
- [SR] It is STRONGLY RECOMMENDED that requirements C-2-1 and C-2-2 above are satisfied by all AAC audio decoders.

When decoding USAC audio, MPEG-D (ISO/IEC 23003-4):

- [C-3-1] Loudness and DRC metadata MUST be interpreted and applied according to MPEG-D DRC Dynamic Range Control Profile Level 1.
- [C-3-2] The decoder MUST behave according to the configuration set with the following android.media.MediaFormat keys: KEY_AAC_DRC_TARGET_REFERENCE_LEVEL and KEY_AAC_DRC_EFFECT_TYPE .

MPEG-4 AAC, HE AAC, and HE AACv2 profile decoders:

- MAY support loudness and dynamic range control using ISO/IEC 23003-4 Dynamic Range Control Profile.

If ISO/IEC 23003-4 is supported and if both ISO/IEC 23003-4 and ISO/IEC 14496-3 metadata are present in a decoded bitstream, then:

- ISO/IEC 23003-4 metadata SHALL take precedence.

All audio decoders MUST support outputting:

- [C-6-1] PCM 16-bit native byte order audio frames via the android.media.MediaCodec API.

5.1.3. Audio Codecs Details

Format/Codec	Details	File Types/Container Formats to be supported
MPEG-4 AAC Profile (AAC LC)	Support for mono/stereo/5.0/5.1 content with standard sampling rates from 8 to 48 kHz.	• 3GPP (.3gp) • MPEG-4 (.mp4, .m4a) • ADTS raw AAC (.aac, ADIF not supported) • MPEG-TS (.ts, not seekable, decode only) • Matroska (.mkv, decode only)
MPEG-4 HE AAC Profile (AAC+)	Support for mono/stereo/5.0/5.1 content with standard sampling rates from 16 to 48 kHz.	• 3GPP (.3gp) • MPEG-4 (.mp4, .m4a)
MPEG-4 HE AACv2 Profile (enhanced AAC+)	Support for mono/stereo/5.0/5.1 content with standard sampling rates from 16 to 48 kHz.	• 3GPP (.3gp) • MPEG-4 (.mp4, .m4a)
AAC ELD (enhanced low delay AAC)	Support for mono/stereo content with standard sampling rates from 16 to 48 kHz.	• 3GPP (.3gp) • MPEG-4 (.mp4, .m4a)
USAC	Support for mono/stereo content with standard sampling rates from 7.35 to 48 kHz.	MPEG-4 (.mp4, .m4a)
AMR-NB	4.75 to 12.2 kbps sampled @ 8 kHz	3GPP (.3gp)
AMR-WB	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16 kHz, as defined at AMR-WB, Adaptive Multi-Rate - Wideband Speech Codec	3GPP (.3gp)
FLAC	For both encoder and decoder: at least Mono and Stereo modes MUST be supported. Sample rates up to 192 kHz MUST be supported; 16-bit and 24-bit resolution MUST be supported. FLAC 24-bit audio data handling MUST be available with floating point audio configuration.	• FLAC (.flac) • MPEG-4 (.mp4, .m4a, decode only) • Matroska (.mkv, decode only)
MP3	Mono/Stereo 8-320Kbps constant (CBR) or variable bitrate (VBR)	• MP3 (.mp3) • MPEG-4 (.mp4, .m4a, decode only) • Matroska (.mkv, decode only)

MIDI	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	• Type 0 and 1 (.mid, .xmf, .mxmf) • RTTTL/RTX (.rtttl, .rtx) • OTA (.ota) • iMelody (.imy)
Vorbis		• Ogg (.ogg) • MPEG-4 (.mp4, .m4a, decode only) • Matroska (.mkv) • Webm (.webm)
PCM/WAVE	PCM codec MUST support 16-bit linear PCM and 16-bit float. WAVE extractor must support 16-bit, 24-bit, 32-bit linear PCM and 32-bit float (rates up to limit of hardware). Sampling rates MUST be supported from 8 kHz to 192 kHz.	WAVE (.wav)
Opus	Decoding: Support for mono, stereo, 5.0 and 5.1 content with sampling rates of 8000, 12000, 16000, 24000, and 48000 Hz. Encoding: Support for mono and stereo content with sampling rates of 8000, 12000, 16000, 24000, and 48000 Hz.	• Ogg (.ogg) • MPEG-4 (.mp4, .m4a, decode only) • Matroska (.mkv) • Webm (.webm)

5.1.4. Image Encoding

See more details in 5.1.6. Image Codecs Details .

Device implementations MUST support encoding the following image encoding:

- [C-0-1] JPEG
- [C-0-2] PNG
- [C-0-3] WebP

If device implementations support HEIC encoding via android.media.MediaCodec for media type MIMETYPE_IMAGE_ANDROID_HEIC , they:

- [C-1-1] MUST provide a hardware-accelerated HEVC encoder codec that supports BITRATE_MODE_CQ bitrate control mode, HEVCProfileMainStill profile and 512 x 512 px frame size.

5.1.5. Image Decoding

See more details in 5.1.6. Image Codecs Details .

Device implementations MUST support decoding the following image encoding:

- [C-0-1] JPEG
- [C-0-2] GIF
- [C-0-3] PNG
- [C-0-4] BMP
- [C-0-5] WebP
- [C-0-6] Raw
- [C-0-7] HEIF (HEIC)

Image decoders that support a high bit-depth format (9+ bits per channel)

- [C-1-1] MUST support outputting an 8-bit equivalent format if requested by the application, for example, via the ARGB_8888 config of android.graphics.Bitmap .

5.1.6. Image Codecs Details

Format/Codec	Details	Supported File Types/Container Formats
JPEG	Base+progressive	JPEG (.jpg)
GIF		GIF (.gif)
PNG		PNG (.png)
BMP		BMP (.bmp)
WebP		WebP (.webp)
Raw		ARW (.arw), CR2 (.cr2), DNG (.dng), NEF (.nef), NRW (.nrw), ORF (.orf), PEF (.pef), RAF (.raf), RW2 (.rw2), SRW (.srw)
HEIF	Image, Image collection, Image sequence	HEIF (.heif), HEIC (.heic)

Image encoder and decoders exposed through the MediaCodec API

- [C-1-1] MUST support YUV420 8:8:8 flexible color format (COLOR_FormatYUV420Flexible) through CodecCapabilities .
- [SR] STRONGLY RECOMMENDED to support RGB888 color format for input Surface mode.
- [C-1-3] MUST support at least one of a planar or semiplanar YUV420 8:8:8 color format: COLOR_FormatYUV420PackedPlanar (equivalent to COLOR_FormatYUV420Planar) or COLOR_FormatYUV420PackedSemiPlanar (equivalent to COLOR_FormatYUV420SemiPlanar). They are STRONGLY RECOMMENDED to support both.

5.1.7. Video Codecs

- For acceptable quality of web video streaming and video-conference services, device implementations SHOULD use a hardware VP8 codec that meets the requirements .

If device implementations include a video decoder or encoder:

- [C-1-1] Video codecs MUST support output and input bytebuffer sizes that accommodate the largest feasible compressed and uncompressed frame as dictated by the standard and configuration but also not overallocate.
- [C-1-2] Video encoders and decoders MUST support YUV420 8:8:8 flexible color formats (COLOR_FormatYUV420Flexible) through CodecCapabilities .
- [C-1-3] Video encoders and decoders MUST support at least one of a planar or semiplanar YUV420 8:8:8 color format: COLOR_FormatYUV420PackedPlanar (equivalent to COLOR_FormatYUV420Planar) or COLOR_FormatYUV420PackedSemiPlanar (equivalent to COLOR_FormatYUV420SemiPlanar). They are STRONGLY RECOMMENDED to support both.
- [SR] Video encoders and decoders are STRONGLY RECOMMENDED to support at least one of a hardware optimized planar or semiplanar YUV420 8:8:8 color format (YV12, NV12, NV21 or equivalent vendor optimized format.)
- [C-1-5] Video decoders that support a high bit-depth format (9+ bits per channel) MUST support outputting an 8-bit equivalent format if requested by the application. This MUST be reflected by supporting an YUV420 8:8:8 color format via android.media.MediaCodecInfo .

If device implementations advertise HDR profile support through Display.HdrCapabilities , they:

- [C-2-1] MUST support HDR static metadata parsing and handling.

If device implementations advertise intra refresh support through FEATURE_IntraRefresh in the MediaCodecInfo.CodecCapabilities class, they:

- [C-3-1] MUST support the refresh periods in the range of 10 - 60 frames and accurately operate within 20% of configured refresh period.

Unless the application specifies otherwise using the KEY_COLOR_FORMAT format key, video decoder implementations:

- [C-4-1] MUST default to the color format optimized for hardware display if configured using Surface output.
- [C-4-2] MUST default to a YUV420 8:8:8 color format optimized for CPU reading if configured to not use Surface output.

5.1.8. Video Codecs List

Format/Codec	Details	File Types/Container Formats to be supported
H.263		• 3GPP (.3gp) • MPEG-4 (.mp4) • Matroska (.mkv, decode only)
H.264 AVC	See section 5.2 and 5.3 for details	• 3GPP (.3gp) • MPEG-4 (.mp4) • MPEG-2 TS (.ts, not seekable) • Matroska (.mkv, decode only)
H.265 HEVC	See section 5.3 for details	• MPEG-4 (.mp4) • Matroska (.mkv, decode only)
MPEG-2	Main Profile	• MPEG2-TS (.ts, not seekable) • MPEG-4 (.mp4, decode only) • Matroska (.mkv, decode only)
MPEG-4 SP		• 3GPP (.3gp) • MPEG-4 (.mp4) • Matroska (.mkv, decode only)
VP8	See section 5.2 and 5.3 for details	• WebM (.webm) • Matroska (.mkv)
VP9	See section 5.3 for details	• WebM (.webm) • Matroska (.mkv)

5.1.9. Media Codec Security

Device implementations MUST ensure compliance with media codec security features as described below.

Android includes support for OMX, a cross-platform multimedia acceleration API, as well as Codec 2.0, a low-overhead multimedia acceleration API.

If device implementations support multimedia, they:

- [C-1-1] MUST provide support for media codecs either via OMX or Codec 2.0 APIs (or both) as in the Android Open Source Project and not disable or circumvent the security protections. This specifically does not mean that every codec MUST use either the OMX or Codec 2.0 API, only that support for at least one of these APIs MUST be available, and support for the available APIs MUST include the security protections present.
- [C-SR] Are STRONGLY RECOMMENDED to include support for Codec 2.0 API.

If device implementations do not support the Codec 2.0 API, they:

- [C-2-1] MUST include the corresponding OMX software codec from the Android Open Source Project (if it is available) for each media format and type (encoder or decoder) supported by the device.

- [C-2-2] Codecs that have names starting with "OMX.google." MUST be based on their Android Open Source Project source code.
- [C-SR] Are STRONGLY RECOMMENDED that the OMX software codecs run in a codec process that does not have access to hardware drivers other than memory mappers.

If device implementations support Codec 2.0 API, they:

- [C-3-1] MUST include the corresponding Codec 2.0 software codec from the Android Open Source Project (if it is available) for each media format and type (encoder or decoder) supported by the device.
- [C-3-2] MUST house the Codec 2.0 software codecs in the software codec process as provided in the Android Open Source Project to make it possible to more narrowly grant access to software codecs.
- [C-3-3] Codecs that have names starting with "c2.android." MUST be based on their Android Open Source Project source code.

5.1.10. Media Codec Characterization

If device implementations support media codecs, they:

- [C-1-1] MUST return correct values of media codec characterization via the MediaCodecInfo API.

In particular:

- [C-1-2] Codecs with names starting with "OMX." MUST use the OMX APIs and have names that conform to OMX IL naming guidelines.
- [C-1-3] Codecs with names starting with "c2." MUST use the Codec 2.0 API and have names that conform to Codec 2.0 naming guidelines for Android.
- [C-1-4] Codecs with names starting with "OMX.google." or "c2.android." MUST NOT be characterized as vendor or as hardware-accelerated.
- [C-1-5] Codecs that run in a codec process (vendor or system) that have access to hardware drivers other than memory allocators and mappers MUST NOT be characterized as software-only.
- [C-1-6] Codecs not present in the Android Open Source Project or not based on the source code in that project MUST be characterized as vendor.
- [C-1-7] Codecs that utilize hardware acceleration MUST be characterized as hardware accelerated.
- [C-1-8] Codec names MUST NOT be misleading. For example, codecs named "decoders" MUST support decoding, and those named "encoders" MUST support encoding. Codecs with names containing media formats MUST support those formats.

If device implementations support video codecs:

- [C-2-1] All video codecs MUST publish achievable frame rate data for the following sizes if supported by the codec:

	SD (low quality)	SD (high quality)	HD 720p	HD 1080p	UHD
Video resolution	• 176 x 144 px (H263, MPEG2, MPEG4) • 352 x 288 px (MPEG4 encoder, H263, MPEG2) • 320 x 180 px (VP8, VP8) • 320 x 240 px (other)	• 704 x 576 px (H263) • 640 x 360 px (VP8, VP9) • 640 x 480 px (MPEG4 encoder) • 720 x 480 px (other)	• 1408 x 1152 px (H263) • 1280 x 720 px (other)	1920 x 1080 px (other than MPEG4)	3840 x 2160 px (HEVC, VP9)

- [C-2-2] Video codecs that are characterized as hardware accelerated MUST publish performance points information. They MUST each list all supported standard performance points (listed in PerformancePoint API), unless they are covered by another supported standard performance point.
- Additionally they SHOULD publish extended performance points if they support sustained

video performance other than one of the standard ones listed.

5.2. Video Encoding

If device implementations support any video encoder and make it available to third-party apps, they:

- SHOULD NOT be, over two sliding windows, more than 15% over the bitrate between intraframe (I-frame) intervals.
- SHOULD NOT be more than 100% over the bitrate over a sliding window of 1 second.

If device implementations include an embedded screen display with the diagonal length of at least 2.5 inches or include a video output port or declare the support of a camera via the android.hardware.camera.any feature flag, they:

- [C-1-1] MUST include the support of at least one of the VP8 or H.264 video encoders, and make it available for third-party applications.
- SHOULD support both VP8 and H.264 video encoders, and make it available for third-party applications.

If device implementations support any of the H.264, VP8, VP9 or HEVC video encoders and make it available to third-party applications, they:

- [C-2-1] MUST support dynamically configurable bitrates.
- SHOULD support variable frame rates, where video encoder SHOULD determine instantaneous frame duration based on the timestamps of input buffers, and allocate its bit bucket based on that frame duration.

If device implementations support the MPEG-4 SP video encoder and make it available to third-party apps, they:

- SHOULD support dynamically configurable bitrates for the supported encoder.

If device implementations provide hardware accelerated video or image encoders, and support one or more attached or pluggable hardware camera(s) exposed through the android.camera APIs:

- [C-4-1] all hardware accelerated video and image encoders MUST support encoding frames from the hardware camera(s).
- SHOULD support encoding frames from the hardware camera(s) through all video or image encoders.

5.2.1. H.263

If device implementations support H.263 encoders and make it available to third-party apps, they:

- [C-1-1] MUST support Baseline Profile Level 45.
- SHOULD support dynamically configurable bitrates for the supported encoder.

5.2.2. H.264

If device implementations support H.264 codec, they:

- [C-1-1] MUST support Baseline Profile Level 3. However, support for ASO (Arbitrary Slice Ordering), FMO (Flexible Macroblock Ordering) and RS (Redundant Slices) is OPTIONAL. Moreover, to maintain compatibility with other Android devices, it is RECOMMENDED that ASO, FMO and RS are not used for Baseline Profile by encoders.
- [C-1-2] MUST support the SD (Standard Definition) video encoding profiles in the following table.
- SHOULD support Main Profile Level 4.
- SHOULD support the HD (High Definition) video encoding profiles as indicated in the following table.

If device implementations report support of H.264 encoding for 720p or 1080p resolution videos through the media APIs, they:

- [C-2-1] MUST support the encoding profiles in the following table.

	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p
Video resolution	320 x 240 px	720 x 480 px	1280 x 720 px	1920 x 1080 px
Video frame rate	20 fps	30 fps	30 fps	30 fps
Video bitrate	384 Kbps	2 Mbps	4 Mbps	10 Mbps

5.2.3. VP8

If device implementations support VP8 codec, they:

- [C-1-1] MUST support the SD video encoding profiles.
- SHOULD support the following HD (High Definition) video encoding profiles.
- [C-1-2] MUST support writing Matroska WebM files.
- SHOULD provide a hardware VP8 codec that meets the WebM project RTC hardware coding requirements , to ensure acceptable quality of web video streaming and video-conference services.

If device implementations report support of VP8 encoding for 720p or 1080p resolution videos through the media APIs, they:

- [C-2-1] MUST support the encoding profiles in the following table.

	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p
Video resolution	320 x 180 px	640 x 360 px	1280 x 720 px	1920 x 1080 px
Video frame rate	30 fps	30 fps	30 fps	30 fps
Video bitrate	800 Kbps	2 Mbps	4 Mbps	10 Mbps

5.2.4. VP9

If device implementations support VP9 codec, they:

- [C-1-2] MUST support Profile 0 Level 3.
- [C-1-1] MUST support writing Matroska WebM files.
- [C-1-3] MUST generate CodecPrivate data.
- SHOULD support the HD decoding profiles as indicated in the following table.
- [SR] are STRONGLY RECOMMENDED to support the HD decoding profiles as indicated in the following table if there is a hardware encoder.

	SD	HD 720p	HD 1080p	UHD
Video resolution	720 x 480 px	1280 x 720 px	1920 x 1080 px	3840 x 2160 px
Video frame rate	30 fps	30 fps	30 fps	30 fps
Video bitrate	1.6 Mbps	4 Mbps	5 Mbps	20 Mbps

If device implementations claim to support Profile 2 or Profile 3 through the Media APIs:

- Support for 12-bit format is OPTIONAL.

5.2.5. H.265

If device implementations support H.265 codec, they:

- [C-1-1] MUST support Main Profile Level 3.
- SHOULD support the HD encoding profiles as indicated in the following table.
- [SR] are STRONGLY RECOMMENDED to support the HD encoding profiles as indicated in the following table if there is a hardware encoder.

	SD	HD 720p	HD 1080p	UHD
Video resolution	720 x 480 px	1280 x 720 px	1920 x 1080 px	3840 x 2160 px
Video frame rate	30 fps	30 fps	30 fps	30 fps

Video bitrate	1.6 Mbps	4 Mbps	5 Mbps	20 Mbps

5.3. Video Decoding

If device implementations support VP8, VP9, H.264, or H.265 codecs, they:

- [C-1-1] MUST support dynamic video resolution and frame rate switching through the standard Android APIs within the same stream for all VP8, VP9, H.264, and H.265 codecs in real time and up to the maximum resolution supported by each codec on the device.

5.3.1. MPEG-2

If device implementations support MPEG-2 decoders, they:

- [C-1-1] MUST support the Main Profile High Level.

5.3.2. H.263

If device implementations support H.263 decoders, they:

- [C-1-1] MUST support Baseline Profile Level 30 and Level 45.

5.3.3. MPEG-4

If device implementations with MPEG-4 decoders, they:

- [C-1-1] MUST support Simple Profile Level 3.

5.3.4. H.264

If device implementations support H.264 decoders, they:

- [C-1-1] MUST support Main Profile Level 3.1 and Baseline Profile. Support for ASO (Arbitrary Slice Ordering), FMO (Flexible Macroblock Ordering) and RS (Redundant Slices) is OPTIONAL.
- [C-1-2] MUST be capable of decoding videos with the SD (Standard Definition) profiles listed in the following table and encoded with the Baseline Profile and Main Profile Level 3.1 (including 720p30).
- SHOULD be capable of decoding videos with the HD (High Definition) profiles as indicated in the following table.

If the height that is reported by the Display.getSupportedModes() method is equal or greater than the video resolution, device implementations:

- [C-2-1] MUST support the HD 720p video decoding profiles in the following table.
- [C-2-2] MUST support the HD 1080p video decoding profiles in the following table.

	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p
Video resolution	320 x 240 px	720 x 480 px	1280 x 720 px	1920 x 1080 px
Video frame rate	30 fps	30 fps	60 fps	30 fps (60 fps [Television])
Video bitrate	800 Kbps	2 Mbps	8 Mbps	20 Mbps

5.3.5. H.265 (HEVC)

If device implementations support H.265 codec, they:

- [C-1-1] MUST support the Main Profile Level 3 Main tier and the SD video decoding profiles as indicated in the following table.
- SHOULD support the HD decoding profiles as indicated in the following table.
- [C-1-2] MUST support the HD decoding profiles as indicated in the following table if there is a hardware decoder.

If the height that is reported by the Display.getSupportedModes() method is equal to or greater than the

video resolution, then:

- [C-2-1] Device implementations MUST support at least one of H.265 or VP9 decoding of 720, 1080 and UHD profiles.

	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p	UHD
Video resolution	352 x 288 px	720 x 480 px	1280 x 720 px	1920 x 1080 px	3840 x 2160 px
Video frame rate	30 fps	30 fps	30 fps	30/60 fps (60 fps Television with H.265 hardware decoding)	60 fps
Video bitrate	600 Kbps	1.6 Mbps	4 Mbps	5 Mbps	20 Mbps

If device implementations claim to support an HDR Profile through the Media APIs:

- [C-3-1] Device implementations MUST accept the required HDR metadata from the application, as well as support extracting and outputting the required HDR metadata from the bitstream and/or container.
- [C-3-2] Device implementations MUST properly display HDR content on the device screen or on a standard video output port (e.g., HDMI).

5.3.6. VP8

If device implementations support VP8 codec, they:

- [C-1-1] MUST support the SD decoding profiles in the following table.
- SHOULD use a hardware VP8 codec that meets the requirements .
- SHOULD support the HD decoding profiles in the following table.

If the height as reported by the Display.getSupportedModes() method is equal or greater than the video resolution, then:

- [C-2-1] Device implementations MUST support 720p profiles in the following table.
- [C-2-2] Device implementations MUST support 1080p profiles in the following table.

	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p
Video resolution	320 x 180 px	640 x 360 px	1280 x 720 px	1920 x 1080 px
Video frame rate	30 fps	30 fps	30 fps (60 fps Television)	30 (60 fps Television)
Video bitrate	800 Kbps	2 Mbps	8 Mbps	20 Mbps

5.3.7. VP9

If device implementations support VP9 codec, they:

- [C-1-1] MUST support the SD video decoding profiles as indicated in the following table.
- SHOULD support the HD decoding profiles as indicated in the following table.

If device implementations support VP9 codec and a hardware decoder:

- [C-2-1] MUST support the HD decoding profiles as indicated in the following table.

If the height that is reported by the Display.getSupportedModes() method is equal to or greater than the video resolution, then:

- [C-3-1] Device implementations MUST support at least one of VP9 or H.265 decoding of the 720, 1080 and UHD profiles.

	SD (Low quality)	SD (High quality)	HD 720p	HD 1080p	UHD
Video resolution	320 x 180 px	640 x 360 px	1280 x 720 px	1920 x 1080 px	3840 x 2160 px

Video frame rate	30 fps	30 fps	30 fps	30 fps (60 fps Television with VP9 hardware decoding)	60 fps
Video bitrate	600 Kbps	1.6 Mbps	4 Mbps	5 Mbps	20 Mbps

If device implementations claim to support VP9Profile2 or VP9Profile3 through the 'CodecProfileLevel' media APIs:

- Support for 12-bit format is OPTIONAL.

If device implementations claim to support an HDR Profile (VP9Profile2HDR , VP9Profile2HDR10Plus , VP9Profile3HDR , VP9Profile3HDR10Plus) through the media APIs:

- [C-4-1] Device implementations MUST accept the required HDR metadata (KEY_HDR_STATIC_INFO for all HDR profiles, as well as 'KEY_HDR10_PLUS_INFO' for HDR10Plus profiles) from the application. They also MUST support extracting and outputting the required HDR metadata from the bitstream and/or container.
- [C-4-2] Device implementations MUST properly display HDR content on the device screen or on a standard video output port (e.g., HDMI).

5.3.8. Dolby Vision

If device implementations declare support for the Dolby Vision decoder through HDR_TYPE_DOLBY_VISION , they:

- [C-1-1] MUST provide a Dolby Vision-capable extractor.
- [C-1-2] MUST properly display Dolby Vision content on the device screen or on a standard video output port (e.g., HDMI).
- [C-1-3] MUST set the track index of backward-compatible base-layer(s) (if present) to be the same as the combined Dolby Vision layer's track index.

5.3.9. AV1

If device implementations support AV1 codec, they:

- [C-1-1] MUST support Profile 0 including 10-bit content.

5.4. Audio Recording

While some of the requirements outlined in this section are listed as SHOULD since Android 4.3, the Compatibility Definition for future versions are planned to change these to MUST. Existing and new Android devices are **STRONGLY RECOMMENDED** to meet these requirements that are listed as SHOULD, or they will not be able to attain Android compatibility when upgraded to the future version.

5.4.1. Raw Audio Capture and Microphone Information

If device implementations declare android.hardware.microphone , they:

- [C-1-1] MUST allow capture of raw audio content with the following characteristics:
  - **Format** : Linear PCM, 16-bit
  - **Sampling rates** : 8000, 11025, 16000, 44100, 48000 Hz
  - **Channels** : Mono
- SHOULD allow capture of raw audio content with the following characteristics:
  - **Format** : Linear PCM, 16-bit and 24-bit
  - **Sampling rates** : 8000, 11025, 16000, 22050, 24000, 32000, 44100, 48000 Hz
  - **Channels** : As many channels as the number of microphones on the device
- [C-1-2] MUST capture at above sample rates without up-sampling.
- [C-1-3] MUST include an appropriate anti-aliasing filter when the sample rates given above are captured with down-sampling.
- SHOULD allow AM radio and DVD quality capture of raw audio content, which means the following characteristics:
  - **Format** : Linear PCM, 16-bit
  - **Sampling rates** : 22050, 48000 Hz

    - **Channels** : Stereo
    - [C-1-4] MUST honor the MicrophoneInfo API and properly fill in information for the available microphones on device accessible to the third-party applications via the AudioManager.getMicrophones() API, and the currently active microphones which are accessible to the third party applications via the AudioRecord.getActiveMicrophones() and MediaRecorder.getActiveMicrophones() APIs. If device implementations allow AM radio and DVD quality capture of raw audio content, they:
- [C-2-1] MUST capture without up-sampling at any ratio higher than 16000:22050 or 44100:48000.
- [C-2-2] MUST include an appropriate anti-aliasing filter for any up-sampling or down-sampling.

5.4.2. Capture for Voice Recognition

If device implementations declare android.hardware.microphone , they:

- [C-1-1] MUST capture android.media.MediaRecorder.AudioSource.VOICE_RECOGNITION audio source at one of the sampling rates, 44100 and 48000.
- [C-1-2] MUST, by default, disable any noise reduction audio processing when recording an audio stream from the AudioSource.VOICE_RECOGNITION audio source.
- [C-1-3] MUST, by default, disable any automatic gain control when recording an audio stream from the AudioSource.VOICE_RECOGNITION audio source.
- SHOULD record the voice recognition audio stream with approximately flat amplitude versus frequency characteristics: specifically, ±3 dB, from 100 Hz to 4000 Hz.
- SHOULD record the voice recognition audio stream with input sensitivity set such that a 90 dB sound power level (SPL) source at 1000 Hz yields RMS of 2500 for 16-bit samples.
- SHOULD record the voice recognition audio stream so that the PCM amplitude levels linearly track input SPL changes over at least a 30 dB range from -18 dB to +12 dB re 90 dB SPL at the microphone.
- SHOULD record the voice recognition audio stream with total harmonic distortion (THD) less than 1% for 1 kHz at 90 dB SPL input level at the microphone.

If device implementations declare android.hardware.microphone and noise suppression (reduction) technologies tuned for speech recognition, they:

- [C-2-1] MUST allow this audio effect to be controllable with the android.media.audiofx.NoiseSuppressor API.
- [C-2-2] MUST uniquely identify each noise suppression technology implementation via the AudioEffect.Descriptor.uuid field.

5.4.3. Capture for Rerouting of Playback

The android.media.MediaRecorder.AudioSource class includes the REMOTE_SUBMIX audio source.
If device implementations declare both android.hardware.audio.output and android.hardware.microphone , they:

- [C-1-1] MUST properly implement the REMOTE_SUBMIX audio source so that when an application uses the android.media.AudioRecord API to record from this audio source, it captures a mix of all audio streams except for the following:
    - AudioManager.STREAM_RING
    - AudioManager.STREAM_ALARM
    - AudioManager.STREAM_NOTIFICATION

5.4.4. Acoustic Echo Canceler

If device implementations declare android.hardware.microphone , they:

- SHOULD implement an Acoustic Echo Canceler (AEC) technology tuned for voice communication and applied to the capture path when capturing using AudioSource.VOICE_COMMUNICATION

If device implementations provides an Acoustic Echo Canceler which is inserted in the capture audio path when AudioSource.VOICE_COMMUNICATION is selected, they:

- [C-SR] are STRONGLY_RECOMMENDED to declare this via AcousticEchoCanceler API method AcousticEchoCanceler.isAvailable()
- [C-SR] are STRONGLY_RECOMMENDED to allow this audio effect to be controllable with the AcousticEchoCanceler API.
- [C-SR] are STRONGLY_RECOMMENDED to uniquely identify each AEC technology implementation via the AudioEffect.Descriptor.uuid field.

5.4.5. Concurrent Capture

If device implementations declare android.hardware.microphone ,they MUST implement concurrent capture as described in this document . Specifically:

- [C-1-1] MUST allow concurrent access to microphone by an accessibility service capturing with AudioSource.VOICE_RECOGNITION and at least one application capturing with any AudioSource .
- [C-1-2] MUST allow concurrent access to microphone by a pre-installed application that holds an Assistant role and at least one application capturing with any AudioSource except for AudioSource.VOICE_COMMUNICATION or AudioSource.CAMCORDER .
- [C-1-3] MUST silence the audio capture for any other application, except for an accessibility service, while an application is capturing with AudioSource.VOICE_COMMUNICATION or AudioSource.CAMCORDER . However, when an app is capturing via AudioSource.VOICE_COMMUNICATION then another app can capture the voice call if it is a privileged (pre-installed) app with permission CAPTURE_AUDIO_OUTPUT .
- [C-1-4] If two or more applications are capturing concurrently and if neither app has an UI on top, the one that started capture the most recently receives audio.

5.4.6. Microphone Gain Levels

If device implementations declare android.hardware.microphone , they:

- SHOULD exhibit approximately flat amplitude-versus-frequency characteristics in the mid-frequency range: specifically ±3dB from 100 Hz to 4000 Hz for each and every microphone used to record the voice recognition audio source.
- SHOULD set audio input sensitivity such that a 1000 Hz sinusoidal tone source played at 90 dB Sound Pressure Level (SPL) yields a response with RMS of 2500 for 16 bit-samples (or -22.35 dB Full Scale for floating point/double precision samples) for each and every microphone used to record the voice recognition audio source.
- [C-SR] are STRONGLY RECOMMENDED to exhibit amplitude levels in the low frequency range: specifically from ±20 dB from 5 Hz to 100 Hz compared to the mid-frequency range for each and every microphone used to record the voice recognition audio source.
- [C-SR] are STRONGLY RECOMMENDED to exhibit amplitude levels in the high frequency range: specifically from ±30 dB from 4000 Hz to 22 KHz compared to the mid-frequency range for each and every microphone used to record the voice recognition audio source.

5.5. Audio Playback

Android includes the support to allow apps to playback audio through the audio output peripheral as defined in section 7.8.2.

5.5.1. Raw Audio Playback

If device implementations declare android.hardware.audio.output , they:

- [C-1-1] MUST allow playback of raw audio content with the following characteristics:
  - **Source formats** : Linear PCM, 16-bit, 8-bit, float
  - **Channels** : Mono, Stereo, valid multichannel configurations with up to 8 channels
  - **Sampling rates (in Hz)** :
    - 8000, 11025, 16000, 22050, 32000, 44100, 48000 at the channel configurations listed above
    - 96000 in mono and stereo
- SHOULD allow playback of raw audio content with the following characteristics:
  - **Sampling rates** : 24000, 48000

5.5.2. Audio Effects

Android provides an API for audio effects for device implementations.

If device implementations declare the feature android.hardware.audio.output , they:

- [C-1-1] MUST support the EFFECT_TYPE_EQUALIZER and EFFECT_TYPE_LOUDNESS_ENHANCER implementations controllable through the AudioEffect subclasses Equalizer and LoudnessEnhancer .
- [C-1-2] MUST support the visualizer API implementation, controllable through the Visualizer class.
- [C-1-3] MUST support the EFFECT_TYPE_DYNAMICS_PROCESSING implementation controllable through the AudioEffect subclass DynamicsProcessing .
- SHOULD support the EFFECT_TYPE_BASS_BOOST , EFFECT_TYPE_ENV_REVERB , EFFECT_TYPE_PRESET_REVERB , and EFFECT_TYPE_VIRTUALIZER implementations controllable through the AudioEffect sub-classes BassBoost , EnvironmentalReverb , PresetReverb , and Virtualizer .
- [C-SR] Are STRONGLY RECOMMENDED to support effects in floating-point and multichannel.

5.5.3. Audio Output Volume

Automotive device implementations:

- SHOULD allow adjusting audio volume separately per each audio stream using the content type or usage as defined by AudioAttributes and car audio usage as publicly defined in android.car.CarAudioManager .

5.6. Audio Latency

Audio latency is the time delay as an audio signal passes through a system. Many classes of applications rely on short latencies, to achieve real-time sound effects.

For the purposes of this section, use the following definitions:

- **output latency** . The interval between when an application writes a frame of PCM-coded data and when the corresponding sound is presented to environment at an on-device transducer or signal leaves the device via a port and can be observed externally.
- **cold output latency** . The output latency for the first frame, when the audio output system has been idle and powered down prior to the request.
- **continuous output latency** . The output latency for subsequent frames, after the device is playing audio.
- **input latency** . The interval between when a sound is presented by environment to device at an on-device transducer or signal enters the device via a port and when an application reads the corresponding frame of PCM-coded data.
- **lost input** . The initial portion of an input signal that is unusable or unavailable.
- **cold input latency** . The sum of lost input time and the input latency for the first frame, when the audio input system has been idle and powered down prior to the request.
- **continuous input latency** . The input latency for subsequent frames, while the device is capturing audio.
- **cold output jitter** . The variability among separate measurements of cold output latency values.
- **cold input jitter** . The variability among separate measurements of cold input latency values.
- **continuous round-trip latency** . The sum of continuous input latency plus continuous output latency plus one buffer period. The buffer period allows time for the app to process the signal and time for the app to mitigate phase difference between input and output streams.
- **OpenSL ES PCM buffer queue API** . The set of PCM-related OpenSL ES APIs within Android NDK .
- **AAudio native audio API** . The set of AAudio APIs within Android NDK .
- **Timestamp** . A pair consisting of a relative frame position within a stream and the estimated time when that frame enters or leaves the audio processing pipeline on the associated endpoint. See also AudioTimestamp .
- **glitch** . A temporary interruption or incorrect sample value in the audio signal, typically caused by a buffer underrun for output, buffer overrun for input, or any other source of

digital or analog noise.

If device implementations declare android.hardware.audio.output , they MUST meet or exceed the following requirements:

- [C-1-1] The output timestamp returned by AudioTrack.getTimestamp and AAudioStream_getTimestamp is accurate to +/- 2 ms.
- [C-1-2] Cold output latency of 500 milliseconds or less.

If device implementations declare android.hardware.audio.output they are STRONGLY RECOMMENDED to meet or exceed the following requirements:

- [C-SR] Cold output latency of 100 milliseconds or less. Existing and new devices that run this version of Android are VERY STRONGLY RECOMMENDED to meet these requirements now. In a future platform release in 2021, we will require Cold output latency of 200 ms or less as a MUST.
- [C-SR] Continuous output latency of 45 milliseconds or less.
- [C-SR] Minimize the cold output jitter.
- [C-SR] The output timestamp returned by AudioTrack.getTimestamp and AAudioStream_getTimestamp is accurate to +/- 1 ms.

If device implementations meet the above requirements, after any initial calibration, when using both the OpenSL ES PCM buffer queue and AAudio native audio APIs, for continuous output latency and cold output latency over at least one supported audio output device, they are:

- [C-SR] STRONGLY RECOMMENDED to report low-latency audio by declaring android.hardware.audio.low_latency feature flag.
- [C-SR] STRONGLY RECOMMENDED to meet the requirements for low-latency audio via the AAudio API.
- [C-SR] STRONGLY RECOMMENDED to ensure that for streams that return AAUDIO_PERFORMANCE_MODE_LOW_LATENCY from AAudioStream_getPerformanceMode() , the value returned by AAudioStream_getFramesPerBurst() is less than or equal to the value returned by android.media.AudioManager.getProperty(String) for property key AudioManager.PROPERTY_OUTPUT_FRAMES_PER_BUFFER .

If device implementations do not meet the requirements for low-latency audio via both the OpenSL ES PCM buffer queue and AAudio native audio APIs, they:

- [C-2-1] MUST NOT report support for low-latency audio.

If device implementations include android.hardware.microphone , they MUST meet these input audio requirements:

- [C-3-1] Limit the error in input timestamps, as returned by AudioRecord.getTimestamp or AAudioStream_getTimestamp , to +/- 2 ms. "Error" here means the deviation from the correct value.
- [C-3-2] Cold input latency of 500 milliseconds or less.

If device implementations include android.hardware.microphone , they are STRONGLY RECOMMENDED to meet these input audio requirements:

- [C-SR] Cold input latency of 100 milliseconds or less. Existing and new devices that run this version of Android are VERY STRONGLY RECOMMENDED to meet these requirements now. In a future platform release in 2021 we will require Cold input latency of 200 ms or less as a MUST.
- [C-SR] Continuous input latency of 30 milliseconds or less.
- [C-SR] Continuous round-trip latency of 50 milliseconds or less.
- [C-SR] Minimize the cold input jitter.
- [C-SR] Limit the error in input timestamps, as returned by AudioRecord.getTimestamp or AAudioStream_getTimestamp , to +/- 1 ms.

5.7. Network Protocols

Device implementations MUST support the media network protocols for audio and video playback as specified in the Android SDK documentation.

If device implementations include an audio or a video decoder, they:

- [C-1-1] MUST support all required codecs and container formats in section 5.1 over HTTP(S).
- [C-1-2] MUST support the media segment formats shown in the Media Segment Formats table below over HTTP Live Streaming draft protocol, Version 7 .
- [C-1-3] MUST support the following RTP audio video profile and related codecs in the RTSP table below. For exceptions please see the table footnotes in section 5.1 .

Media Segment Formats

Segment formats	Reference(s)	Required codec support
MPEG-2 Transport Stream	ISO 13818	Video codecs: • H264 AVC • MPEG-4 SP • MPEG-2 See section 5.1.3 for details on H264 AVC, MPEG2-4 SP, and MPEG-2. Audio codecs: • AAC See section 5.1.1 for details on AAC and its variants.
AAC with ADTS framing and ID3 tags	ISO 13818-7	See section 5.1.1 for details on AAC and its variants
WebVTT	WebVTT	

RTSP (RTP, SDP)

Profile name	Reference(s)	Required codec support
H264 AVC	RFC 6184	See section 5.1.3 for details on H264 AVC
MP4A-LATM	RFC 6416	See section 5.1.1 for details on AAC and its variants
H263-1998	RFC 3551 RFC 4629 RFC 2190	See section 5.1.3 for details on H263
H263-2000	RFC 4629	See section 5.1.3 for details on H263
AMR	RFC 4867	See section 5.1.1 for details on AMR-NB
AMR-WB	RFC 4867	See section 5.1.1 for details on AMR-WB
MP4V-ES	RFC 6416	See section 5.1.3 for details on MPEG-4 SP
mpeg4-generic	RFC 3640	See section 5.1.1 for details on AAC and its variants
MP2T	RFC 2250	See MPEG-2 Transport Stream underneath HTTP Live Streaming for details

5.8. Secure Media

If device implementations support secure video output and are capable of supporting secure surfaces, they:

- [C-1-1] MUST declare support for Display.FLAG_SECURE .

If device implementations declare support for Display.FLAG_SECURE and support wireless display protocol, they:

- [C-2-1] MUST secure the link with a cryptographically strong mechanism such as HDCP 2.x or higher for the displays connected through wireless protocols such as Miracast.

If device implementations declare support for Display.FLAG_SECURE and support wired external display, they:

- [C-3-1] MUST support HDCP 1.2 or higher for all external displays connected via a user-accessible wired port.

5.9. Musical Instrument Digital Interface (MIDI)

If device implementations report support for feature android.software.midi via the android.content.pm.PackageManager class, they:

- [C-1-1] MUST support MIDI over *all* MIDI-capable hardware transports for which they provide generic non-MIDI connectivity, where such transports are:
  - USB host mode, section 7.7
  - MIDI over Bluetooth LE acting in central role, section 7.4.3
- [C-1-2] MUST support the inter-app MIDI software transport (virtual MIDI devices)
- [C-1-3] MUST include libamidi.so (native MIDI support)
- SHOULD support MIDI over USB peripheral mode, section 7.7

5.10. Professional Audio

If device implementations report support for feature android.hardware.audio.pro via the android.content.pm.PackageManager class, they:

- [C-1-1] MUST report support for feature android.hardware.audio.low_latency .
- [C-1-2] MUST have the continuous round-trip audio latency, as defined in section 5.6 Audio Latency , MUST be 20 milliseconds or less and SHOULD be 10 milliseconds or less over at least one supported path.
- [C-1-3] MUST include a USB port(s) supporting USB host mode and USB peripheral mode.
- [C-1-4] MUST report support for feature android.software.midi .
- [C-1-5] MUST meet latencies and USB audio requirements using both the OpenSL ES PCM buffer queue API and at least one path of the AAudio native audio API.
- [SR] Are STRONGLY RECOMMENDED to meet latencies and USB audio requirements using the AAudio native audio API over the MMAP path .
- [C-1-6] MUST have Cold output latency of 200 milliseconds or less.
- [C-1-7] MUST have Cold input latency of 200 milliseconds or less.
- [SR] Are STRONGLY RECOMMENDED to provide a consistent level of CPU performance while audio is active and CPU load is varying. This should be tested using the Android app version of SynthMark commit id 09b13c6f49ea089f8c31e5d035f912cc405b7ab8 . SynthMark uses a software synthesizer running on a simulated audio framework that measures system performance. The SynthMark app needs to be run using the “Automated Test” option and achieve the following results:
  - voicemark.90 >= 32 voices
  - latencymark.fixed.little <= 15 msec
  - latencymark.dynamic.little <= 50 msec

See the SynthMark documentation for an explanation of the benchmarks.

- SHOULD minimize audio clock inaccuracy and drift relative to standard time.
- SHOULD minimize audio clock drift relative to the CPU CLOCK_MONOTONIC when both are active.
- SHOULD minimize audio latency over on-device transducers.
- SHOULD minimize audio latency over USB digital audio.
- SHOULD document audio latency measurements over all paths.
- SHOULD minimize jitter in audio buffer completion callback entry times, as this affects usable percentage of full CPU bandwidth by the callback.
- SHOULD provide zero audio glitches under normal use at reported latency.
- SHOULD provide zero inter-channel latency difference.
- SHOULD minimize MIDI mean latency over all transports.
- SHOULD minimize MIDI latency variability under load (jitter) over all transports.
- SHOULD provide accurate MIDI timestamps over all transports.
- SHOULD minimize audio signal noise over on-device transducers, including the period

immediately after cold start.

- SHOULD provide zero audio clock difference between the input and output sides of corresponding end-points, when both are active. Examples of corresponding end-points include the on-device microphone and speaker, or the audio jack input and output.
- SHOULD handle audio buffer completion callbacks for the input and output sides of corresponding end-points on the same thread when both are active, and enter the output callback immediately after the return from the input callback. Or if it is not feasible to handle the callbacks on the same thread, then enter the output callback shortly after entering the input callback to permit the application to have a consistent timing of the input and output sides.
- SHOULD minimize the phase difference between HAL audio buffering for the input and output sides of corresponding end-points.
- SHOULD minimize touch latency.
- SHOULD minimize touch latency variability under load (jitter).
- SHOULD have a latency from touch input to audio output of less than or equal to 40 ms.

If device implementations meet all of the above requirements, they:

- [SR] STRONGLY RECOMMENDED to report support for feature android.hardware.audio.pro via the android.content.pm.PackageManager class.

If device implementations include a 4 conductor 3.5mm audio jack, they:

- [C-2-1] MUST have the continuous round-trip audio latency to be 20 milliseconds or less over the audio jack path.
- [SR] STRONGLY RECOMMENDED to comply with section Mobile device (jack) specifications of the Wired Audio Headset Specification (v1.1) .
- The continuous round-trip audio latency SHOULD be 10 milliseconds or less over the audio jack path.

If device implementations omit a 4 conductor 3.5mm audio jack and include a USB port(s) supporting USB host mode, they:

- [C-3-1] MUST implement the USB audio class.
- [C-3-2] MUST have a continuous round-trip audio latency of 20 milliseconds or less over the USB host mode port using USB audio class.
- The continuous round-trip audio latency SHOULD be 10 milliseconds or less over the USB host mode port using USB audio class.
- [C-SR] Are STRONGLY RECOMMENDED to support simultaneous I/O up to 8 channels each direction, 96 kHz sample rate, and 24-bit or 32-bit depth, when used with USB audio peripherals that also support these requirements.

If device implementations include an HDMI port, they:

- SHOULD support output in stereo and eight channels at 20-bit or 24-bit depth and 192 kHz without bit-depth loss or resampling, in at least one configuration.

5.11. Capture for Unprocessed

Android includes support for recording of unprocessed audio via the android.media.MediaRecorder.AudioSource.UNPROCESSED audio source. In OpenSL ES, it can be accessed with the record preset SL_ANDROID_RECORDING_PRESET_UNPROCESSED .

If device implementations intent to support unprocessed audio source and make it available to third-party apps, they:

- [C-1-1] MUST report the support through the android.media.AudioManager property PROPERTY_SUPPORT_AUDIO_SOURCE_UNPROCESSED .
- [C-1-2] MUST exhibit approximately flat amplitude-versus-frequency characteristics in the mid-frequency range: specifically ±10dB from 100 Hz to 7000 Hz for each and every microphone used to record the unprocessed audio source.
- [C-1-3] MUST exhibit amplitude levels in the low frequency range: specifically from ±20 dB from 5 Hz to 100 Hz compared to the mid-frequency range for each and every microphone used to record the unprocessed audio source.
- [C-1-4] MUST exhibit amplitude levels in the high frequency range: specifically from ±30

dB from 7000 Hz to 22 KHz compared to the mid-frequency range for each and every microphone used to record the unprocessed audio source.

- [C-1-5] MUST set audio input sensitivity such that a 1000 Hz sinusoidal tone source played at 94 dB Sound Pressure Level (SPL) yields a response with RMS of 520 for 16 bit-samples (or -36 dB Full Scale for floating point/double precision samples) for each and every microphone used to record the unprocessed audio source.
- [C-1-6] MUST have a signal-to-noise ratio (SNR) at 60 dB or higher for each and every microphone used to record the unprocessed audio source. (whereas the SNR is measured as the difference between 94 dB SPL and equivalent SPL of self noise, A-weighted).
- [C-1-7] MUST have a total harmonic distortion (THD) less than be less than 1% for 1 kHZ at 90 dB SPL input level at each and every microphone used to record the unprocessed audio source.
- MUST not have any other signal processing (e.g. Automatic Gain Control, High Pass Filter, or Echo cancellation) in the path other than a level multiplier to bring the level to desired range. In other words:
- [C-1-8] If any signal processing is present in the architecture for any reason, it MUST be disabled and effectively introduce zero delay or extra latency to the signal path.
- [C-1-9] The level multiplier, while allowed to be on the path, MUST NOT introduce delay or latency to the signal path.

All SPL measurements are made directly next to the microphone under test. For multiple microphone configurations, these requirements apply to each microphone.

If device implementations declare `android.hardware.microphone` but do not support unprocessed audio source, they:

- [C-2-1] MUST return `null` for the `AudioManager.getProperty(PROPERTY_SUPPORT_AUDIO_SOURCE_UNPROCESSED)` API method, to properly indicate the lack of support.
- [SR] are still STRONGLY RECOMMENDED to satisfy as many of the requirements for the signal path for the unprocessed recording source.

6. Developer Tools and Options Compatibility

6.1. Developer Tools

Device implementations:

- [C-0-1] MUST support the Android Developer Tools provided in the Android SDK.
- Android Debug Bridge (adb)
  - [C-0-2] MUST support adb as documented in the Android SDK and the shell commands provided in the AOSP, which can be used by app developers, including dumpsys `cmd stats`
  - [C-0-11] MUST support the shell command `cmd testharness` . Upgrading device implementations from an earlier Android version without a persistent data block MAY be exempted from C-0-11.
  - [C-0-3] MUST NOT alter the format or the contents of device system events (batterystats , diskstats, fingerprint, graphicsstats, netstats, notification, procstats) logged via the dumpsys command.
  - [C-0-10] MUST record, without omission, and make the following events accessible and available to the `cmd stats` shell command and the `StatsManager` System API class.
    - ActivityForegroundStateChanged
    - AnomalyDetected
    - AppBreadcrumbReported
    - AppCrashOccurred
    - AppStartOccurred
    - BatteryLevelChanged
    - BatterySaverModeStateChanged
    - BleScanResultReceived
    - BleScanStateChanged
    - ChargingStateChanged
    - DeviceIdleModeStateChanged

        - ForegroundServiceStateChanged
        - GpsScanStateChanged
        - JobStateChanged
        - PluggedStateChanged
        - ScheduledJobStateChanged
        - ScreenStateChanged
        - SyncStateChanged
        - SystemElapsedRealtime
        - UidProcessStateChanged
        - WakelockStateChanged
        - WakeupAlarmOccurred
        - WifiLockStateChanged
        - WifiMulticastLockStateChanged
        - WifiScanStateChanged
    - [C-0-4] MUST have the device-side adb daemon be inactive by default and there MUST be a user-accessible mechanism to turn on the Android Debug Bridge.
    - [C-0-5] MUST support secure adb. Android includes support for secure adb. Secure adb enables adb on known authenticated hosts.
    - [C-0-6] MUST provide a mechanism allowing adb to be connected from a host machine. Specifically:

  If device implementations without a USB port support peripheral mode, they:
    - [C-3-1] MUST implement adb via local-area network (such as Ethernet or Wi-Fi).
    - [C-3-2] MUST provide drivers for Windows 7, 8 and 10, allowing developers to connect to the device using the adb protocol.

  If device implementations support adb connections to a host machine via Wi-Fi, they:
    - [C-4-1] MUST have the `AdbManager#isAdbWifiSupported()` method return `true` .

  If device implementations support adb connections to a host machine via Wi-Fi and includes at least one camera, they:
    - [C-5-1] MUST have the `AdbManager#isAdbWifiQrSupported()` method return `true` .
- Dalvik Debug Monitor Service (ddms)
    - [C-0-7] MUST support all ddms features as documented in the Android SDK. As ddms uses adb, support for ddms SHOULD be inactive by default, but MUST be supported whenever the user has activated the Android Debug Bridge, as above.
- Monkey
    - [C-0-8] MUST include the Monkey framework and make it available for applications to use.
- SysTrace
    - [C-0-9] MUST support the systrace tool as documented in the Android SDK. Systrace must be inactive by default and there MUST be a user-accessible mechanism to turn on Systrace.
- Perfetto
    - [C-SR] Are STRONGLY RECOMMENDED to expose a `/system/bin/perfetto` binary to the shell user which cmdline complies with the perfetto documentation .
    - [C-SR] The perfetto binary is STRONGLY RECOMMENDED to accept as input a protobuf config that complies with the schema defined in the perfetto documentation .
    - [C-SR] The perfetto binary is STRONGLY RECOMMENDED to write as output a protobuf trace that complies with the schema defined in the perfetto documentation .
    - [C-SR] Are STRONGLY RECOMMENDED to provide, through the perfetto binary, at least the data sources described in the perfetto documentation .
- Low Memory Killer
    - [C-0-10] MUST write a `LMK_KILL_OCCURRED_FIELD_NUMBER` Atom to the statsd log when an app is terminated by the Low Memory Killer .
- Test Harness Mode If device implementations support the shell command `cmd testharness` and run `cmd testharness enable` , they:
    - [C-2-1] MUST return `true` for `ActivityManager.isRunningInUserTestHarness()`
    - [C-2-2] MUST implement Test Harness Mode as described in Test Harness

Mode documentation .

If device implementations report the support of Vulkan 1.0 or higher via the android.hardware.vulkan.version feature flags, they:

- [C-1-1] MUST provide an affordance for the app developer to enable/disable GPU debug layers.
- [C-1-2] MUST, when the GPU debug layers are enabled, enumerate layers in libraries provided by external tools (i.e. not part of the platform or application package) found in debuggable applications' base directory to support vkEnumerateInstanceLayerProperties() and vkCreateInstance() API methods.

6.2. Developer Options

Android includes support for developers to configure application development-related settings. Device implementations MUST provide a consistent experience for Developer Options, they:

- [C-0-1] MUST honor the android.settings.APPLICATION_DEVELOPMENT_SETTINGS intent to show application development-related settings. The upstream Android implementation hides the Developer Options menu by default and enables users to launch Developer Options after pressing seven (7) times on the **Settings** > **About Device** > **Build Number** menu item.
- [C-0-2] MUST hide Developer Options by default.
- [C-0-3] MUST provide a clear mechanism that does not give preferential treatment to one third-party app as opposed to another to enable Developer Options. MUST provide a public visible document or website that describes how to enable Developer Options. This document or website MUST be linkable from the Android SDK documents.
- SHOULD have an ongoing visual notification to the user when Developer Options is enabled and the safety of the user is of concern.
- MAY temporarily limit access to the Developer Options menu, by visually hiding or disabling the menu, to prevent distraction for scenarios where the safety of the user is of concern.

7. Hardware Compatibility

If a device includes a particular hardware component that has a corresponding API for third-party developers:

- [C-0-1] The device implementation MUST implement that API as described in the Android SDK documentation.

If an API in the SDK interacts with a hardware component that is stated to be optional and the device implementation does not possess that component:

- [C-0-2] Complete class definitions (as documented by the SDK) for the component APIs MUST still be presented.
- [C-0-3] The API's behaviors MUST be implemented as no-ops in some reasonable fashion.
- [C-0-4] API methods MUST return null values where permitted by the SDK documentation.
- [C-0-5] API methods MUST return no-op implementations of classes where null values are not permitted by the SDK documentation.
- [C-0-6] API methods MUST NOT throw exceptions not documented by the SDK documentation.
- [C-0-7] Device implementations MUST consistently report accurate hardware configuration information via the getSystemAvailableFeatures() and hasSystemFeature(String) methods on the android.content.pm.PackageManager class for the same build fingerprint.

A typical example of a scenario where these requirements apply is the telephony API: Even on non-phone devices, these APIs must be implemented as reasonable no-ops.

7.1. Display and Graphics

Android includes facilities that automatically adjust application assets and UI layouts appropriately for the device to ensure that third-party applications run well on a variety of hardware configurations . On the Android-compatible display(s) where all third-party Android-compatible applications can run,

device implementations MUST properly implement these APIs and behaviors, as detailed in this section.

The units referenced by the requirements in this section are defined as follows:

- **physical diagonal size** . The distance in inches between two opposing corners of the illuminated portion of the display.
- **dots per inch (dpi)** . The number of pixels encompassed by a linear horizontal or vertical span of 1". Where dpi values are listed, both horizontal and vertical dpi must fall within the range.
- **aspect ratio** . The ratio of the pixels of the longer dimension to the shorter dimension of the screen. For example, a display of 480x854 pixels would be 854/480 = 1.779, or roughly "16:9".
- **density-independent pixel (dp)** . The virtual pixel unit normalized to a 160 dpi screen, calculated as: pixels = dps * (density/160).

7.1.1. Screen Configuration

7.1.1.1. Screen Size and Shape

The Android UI framework supports a variety of different logical screen layout sizes, and allows applications to query the current configuration's screen layout size via Configuration.screenLayout with the SCREENLAYOUT_SIZE_MASK and Configuration.smallestScreenWidthDp .

Device implementations:

- [C-0-1] MUST report the correct layout size for the Configuration.screenLayout as defined in the Android SDK documentation. Specifically, device implementations MUST report the correct logical density-independent pixel (dp) screen dimensions as below:
  - Devices with the Configuration.uiMode set as any value other than UI_MODE_TYPE_WATCH, and reporting a small size for the Configuration.screenLayout , MUST have at least 426 dp x 320 dp.
  - Devices reporting a normal size for the Configuration.screenLayout , MUST have at least 480 dp x 320 dp.
  - Devices reporting a large size for the Configuration.screenLayout , MUST have at least 640 dp x 480 dp.
  - Devices reporting a xlarge size for the Configuration.screenLayout , MUST have at least 960 dp x 720 dp.
- [C-0-2] MUST correctly honor applications' stated support for screen sizes through the < supports-screens > attribute in the AndroidManifest.xml, as described in the Android SDK documentation.
- MAY have the Android-compatible display(s) with rounded corners.

If device implementations support UI_MODE_TYPE_NORMAL and include Android-compatible display(s) with rounded corners, they:

- [C-1-1] MUST ensure that at least one of the following requirements is met:
- The radius of the rounded corners is less than or equal to 38 dp.
- When a 15 dp by 15 dp box is anchored at each corner of the logical display, at least one pixel of each box is visible on the screen.
- SHOULD include user affordance to switch to the display mode with the rectangular corners.

If device implementations include an Android-compatible display(s) that is foldable, or includes a folding hinge between multiple display panels and makes such display(s) available to render third-party apps, they:

- [C-2-1] MUST implement the latest available stable version of the extensions API or the stable version of sidecar API to be used by Window Manager Jetpack library.

If device implementations include an Android-compatible display(s) that is foldable, or includes a folding hinge between multiple display panels and if the hinge or fold crosses a fullscreen application window, they:

- [C-3-1] MUST report the position, bounds and state of hinge or fold through extensions or sidecar APIs to the application.

For details on correctly implementing the sidecar or extension APIs refer to the public documentation of Window Manager Jetpack .

7.1.1.2. Screen Aspect Ratio

While there is no restriction to the aspect ratio of the physical display for the Android-compatible display(s), the aspect ratio of the logical display where third-party apps are rendered, which can be derived from the height and width values reported through the view.Display APIs and Configuration APIs, MUST meet the following requirements:

- [C-0-1] Device implementations with Configuration.uiMode set to UI_MODE_TYPE_NORMAL MUST have an aspect ratio value less than or equal to 1.86 (roughly 16:9), unless the app meets one of the following conditions:
  - The app has declared that it supports a larger screen aspect ratio through the android.max_aspect metadata value.
  - The app declares it is resizeable via the android:resizeableActivity attribute.
  - The app targets API level 24 or higher and does not declare an android:maxAspectRatio that would restrict the allowed aspect ratio.
- [C-0-2] Device implementations with Configuration.uiMode set to UI_MODE_TYPE_NORMAL MUST have an aspect ratio value equal to or greater than 1.3333 (4:3), unless the app can be stretched wider by meeting one of the following conditions:
  - The app declares it is resizeable via the android:resizeableActivity attribute.
  - The app declares an android:minAspectRatio that would restrict the allowed aspect ratio.
- [C-0-3] Device implementations with the Configuration.uiMode set as UI_MODE_TYPE_WATCH MUST have an aspect ratio value set as 1.0 (1:1).

7.1.1.3. Screen Density

The Android UI framework defines a set of standard logical densities to help application developers target application resources.

- [C-0-1] By default, device implementations MUST report only one of the following logical Android framework densities through the DENSITY_DEVICE_STABLE API and this value MUST NOT change at any time; however, the device MAY report a different arbitrary density according to the display configuration changes made by the user (for example, display size) set after initial boot.
  - 120 dpi (ldpi)
  - 140 dpi (140dpi)
  - 160 dpi (mdpi)
  - 180 dpi (180dpi)
  - 200 dpi (200dpi)
  - 213 dpi (tvdpi)
  - 220 dpi (220dpi)
  - 240 dpi (hdpi)
  - 260 dpi (260dpi)
  - 280 dpi (280dpi)
  - 300 dpi (300dpi)
  - 320 dpi (xhdpi)
  - 340 dpi (340dpi)
  - 360 dpi (360dpi)
  - 400 dpi (400dpi)
  - 420 dpi (420dpi)
  - 480 dpi (xxhdpi)
  - 560 dpi (560dpi)
  - 640 dpi (xxxhdpi)
- Device implementations SHOULD define the standard Android framework density that is numerically closest to the physical density of the screen, unless that logical density pushes the reported screen size below the minimum supported. If the standard Android framework density that is numerically closest to the physical density results in a screen

size that is smaller than the smallest supported compatible screen size (320 dp width), device implementations SHOULD report the next lowest standard Android framework density.

If there is an affordance to change the display size of the device:

- [C-1-1] The display size MUST NOT be scaled any larger than 1.5 times the native density or produce an effective minimum screen dimension smaller than 320dp (equivalent to resource qualifier sw320dp), whichever comes first.
- [C-1-2] Display size MUST NOT be scaled any smaller than 0.85 times the native density.
- To ensure good usability and consistent font sizes, it is RECOMMENDED that the following scaling of Native Display options be provided (while complying with the limits specified above)
- Small: 0.85x
- Default: 1x (Native display scale)
- Large: 1.15x
- Larger: 1.3x
- Largest 1.45x

7.1.2. Display Metrics

If device implementations include the Android-compatible display(s) or video output to the Android-compatible display screen(s), they:

- [C-1-1] MUST report correct values for all Android-compatible display metrics defined in the android.util.DisplayMetrics API.

If device implementations does not include an embedded screen or video output, they:

- [C-2-1] MUST report correct values of the Android-compatible display as defined in the android.util.DisplayMetrics API for the emulated default view.Display .

7.1.3. Screen Orientation

Device implementations:

- [C-0-1] MUST report which screen orientations they support (android.hardware.screen.portrait and/or android.hardware.screen.landscape) and MUST report at least one supported orientation. For example, a device with a fixed orientation landscape screen, such as a television or laptop, SHOULD only report android.hardware.screen.landscape .
- [C-0-2] MUST report the correct value for the device's current orientation, whenever queried via the android.content.res.Configuration.orientation , android.view.Display.getOrientation() , or other APIs.

If device implementations support both screen orientations, they:

- [C-1-1] MUST support dynamic orientation by applications to either portrait or landscape screen orientation. That is, the device must respect the application's request for a specific screen orientation.
- [C-1-2] MUST NOT change the reported screen size or density when changing orientation.
- MAY select either portrait or landscape orientation as the default.

7.1.4. 2D and 3D Graphics Acceleration

7.1.4.1 OpenGL ES

Device implementations:

- [C-0-1] MUST correctly identify the supported OpenGL ES versions (1.1, 2.0, 3.0, 3.1, 3.2) through the managed APIs (such as via the GLES10.getString() method) and the native APIs.
- [C-0-2] MUST include the support for all the corresponding managed APIs and native APIs for every OpenGL ES versions they identified to support.

If device implementations include a screen or video output, they:

- [C-1-1] MUST support both OpenGL ES 1.1 and 2.0, as embodied and detailed in the Android SDK documentation .
- [C-SR] Are STRONGLY RECOMMENDED to support OpenGL ES 3.1.
- SHOULD support OpenGL ES 3.2.

If device implementations support any of the OpenGL ES versions, they:

- [C-2-1] MUST report via the OpenGL ES managed APIs and native APIs any other OpenGL ES extensions they have implemented, and conversely MUST NOT report extension strings that they do not support.
- [C-2-2] MUST support the EGL_KHR_image , EGL_KHR_image_base , EGL_ANDROID_image_native_buffer , EGL_ANDROID_get_native_client_buffer , EGL_KHR_wait_sync , EGL_KHR_get_all_proc_addresses , EGL_ANDROID_presentation_time , EGL_KHR_swap_buffers_with_damage , EGL_ANDROID_recordable , and EGL_ANDROID_GLES_layers extensions.
- [C-SR] Are STRONGLY RECOMMENDED to support the EGL_KHR_partial_update and OES_EGL_image_external extensions.
- SHOULD accurately report via the getString() method, any texture compression format that they support, which is typically vendor-specific.

If device implementations declare support for OpenGL ES 3.0, 3.1, or 3.2, they:

- [C-3-1] MUST export the corresponding function symbols for these version in addition to the OpenGL ES 2.0 function symbols in the libGLESv2.so library.
- [SR] Are STRONGLY RECOMMENDED to support the OES_EGL_image_external_essl3 extension.

If device implementations support OpenGL ES 3.2, they:

- [C-4-1] MUST support the OpenGL ES Android Extension Pack in its entirety.

If device implementations support the OpenGL ES Android Extension Pack in its entirety, they:

- [C-5-1] MUST identify the support through the android.hardware.opengles.aep feature flag.

If device implementations expose support for the EGL_KHR_mutable_render_buffer extension, they:

- [C-6-1] MUST also support the EGL_ANDROID_front_buffer_auto_refresh extension.

7.1.4.2 Vulkan

Android includes support for Vulkan , a low-overhead, cross-platform API for high-performance 3D graphics.

If device implementations support OpenGL ES 3.1, they:

- [SR] Are STRONGLY RECOMMENDED to include support for Vulkan 1.1.

If device implementations include a screen or video output, they:

- SHOULD include support for Vulkan 1.1.

The Vulkan dEQP tests are partitioned into a number of test lists, each with an associated date/version. These are in the Android source tree at external/deqp/android/cts/master/vk-master-YYYY-MM-DD.txt . A device that supports Vulkan at a self-reported level indicates that it can pass the dEQP tests in all test lists from this level and earlier.

If device implementations include support for Vulkan 1.0 or higher, they:

- [C-1-1] MUST report the correct integer value with the android.hardware.vulkan.level and android.hardware.vulkan.version feature flags.
- [C-1-2] MUST enumerate, at least one VkPhysicalDevice for the Vulkan native API vkEnumeratePhysicalDevices() .
- [C-1-3] MUST fully implement the Vulkan 1.0 APIs for each enumerated VkPhysicalDevice .
- [C-1-4] MUST enumerate layers, contained in native libraries named as libVkLayer*.so in

the application package's native library directory, through the Vulkan native APIs vkEnumerateInstanceLayerProperties() and vkEnumerateDeviceLayerProperties() .

- [C-1-5] MUST NOT enumerate layers provided by libraries outside of the application package, or provide other ways of tracing or intercepting the Vulkan API, unless the application has the android:debuggable attribute set as true .
- [C-1-6] MUST report all extension strings that they do support via the Vulkan native APIs , and conversely MUST NOT report extension strings that they do not correctly support.
- [C-1-7] MUST support the VK_KHR_surface, VK_KHR_android_surface, VK_KHR_swapchain, and VK_KHR_incremental_present extensions.
- [C-1-8] MUST report the maximum version of the Vulkan dEQP Tests supported via the android.software.vulkan.deqp.level feature flag.
- [C-1-9] MUST at least support version 132317953 (from Mar 1st, 2019) as reported in the android.software.vulkan.deqp.level feature flag.
- [C-1-10] MUST pass all Vulkan dEQP Tests in the test lists between version 132317953 and the version specified in the android.software.vulkan.deqp.level feature flag.
- [C-SR] Are STRONGLY RECOMMENDED to support the VK_KHR_driver_properties and VK_GOOGLE_display_timing extensions.

If device implementations do not include support for Vulkan 1.0, they:

- [C-2-1] MUST NOT declare any of the Vulkan feature flags (e.g. android.hardware.vulkan.level , android.hardware.vulkan.version).
- [C-2-2] MUST NOT enumerate any VkPhysicalDevice for the Vulkan native API vkEnumeratePhysicalDevices() .

If device implementations include support for Vulkan 1.1 and declare any of the Vulkan feature flags, they:

- [C-3-1] MUST expose support for the SYNC_FD external semaphore and handle types and the VK_ANDROID_external_memory_android_hardware_buffer extension.

7.1.4.3 RenderScript

- [C-0-1] Device implementations MUST support Android RenderScript , as detailed in the Android SDK documentation.

7.1.4.4 2D Graphics Acceleration

Android includes a mechanism for applications to declare that they want to enable hardware acceleration for 2D graphics at the Application, Activity, Window, or View level through the use of a manifest tag android:hardwareAccelerated or direct API calls.

Device implementations:

- [C-0-1] MUST enable hardware acceleration by default, and MUST disable hardware acceleration if the developer so requests by setting android:hardwareAccelerated="false" or disabling hardware acceleration directly through the Android View APIs.
- [C-0-2] MUST exhibit behavior consistent with the Android SDK documentation on hardware acceleration .

Android includes a TextureView object that lets developers directly integrate hardware-accelerated OpenGL ES textures as rendering targets in a UI hierarchy.

Device implementations:

- [C-0-3] MUST support the TextureView API, and MUST exhibit consistent behavior with the upstream Android implementation.

7.1.4.5 Wide-gamut Displays

If device implementations claim support for wide-gamut displays through Configuration.isScreenWideColorGamut() , they:

- [C-1-1] MUST have a color-calibrated display.
- [C-1-2] MUST have a display whose gamut covers the sRGB color gamut entirely in CIE 1931 xyY space.

- [C-1-3] MUST have a display whose gamut has an area of at least 90% of DCI-P3 in CIE 1931 xyY space.
- [C-1-4] MUST support OpenGL ES 3.1 or 3.2 and report it properly.
- [C-1-5] MUST advertise support for the EGL_KHR_no_config_context , EGL_EXT_pixel_format_float , EGL_KHR_gl_colorspace , EGL_EXT_gl_colorspace_scrgb , EGL_EXT_gl_colorspace_scrgb_linear , EGL_EXT_gl_colorspace_display_p3 , EGL_EXT_gl_colorspace_display_p3_linear , and EGL_EXT_gl_colorspace_display_p3_passthrough extensions.
- [C-SR] Are STRONGLY RECOMMENDED to support GL_EXT_sRGB .

Conversely, if device implementations do not support wide-gamut displays, they:

- [C-2-1] SHOULD cover 100% or more of sRGB in CIE 1931 xyY space, although the screen color gamut is undefined.

7.1.5. Legacy Application Compatibility Mode

Android specifies a “compatibility mode” in which the framework operates in a 'normal' screen size equivalent (320dp width) mode for the benefit of legacy applications not developed for old versions of Android that pre-date screen-size independence.

7.1.6. Screen Technology

The Android platform includes APIs that allow applications to render rich graphics to an Android-compatible display. Devices MUST support all of these APIs as defined by the Android SDK unless specifically allowed in this document.

All of a device implementation's Android-compatible displays:

- [C-0-1] MUST be capable of rendering 16-bit color graphics.
- SHOULD support displays capable of 24-bit color graphics.
- [C-0-2] MUST be capable of rendering animations.
- [C-0-3] MUST have a pixel aspect ratio (PAR) between 0.9 and 1.15. That is, the pixel aspect ratio MUST be near square (1.0) with a 10 ~ 15% tolerance.

7.1.7. Secondary Displays

Android includes support for secondary Android-compatible displays to enable media sharing capabilities and developer APIs for accessing external displays.

If device implementations support an external display either via a wired, wireless, or an embedded additional display connection, they:

- [C-1-1] MUST implement the DisplayManager system service and API as described in the Android SDK documentation.

7.2. Input Devices

Device implementations:

- [C-0-1] MUST include an input mechanism, such as atouchscreen or non-touch navigation , to navigate between the UI elements.

7.2.1. Keyboard

If device implementations include support for third-party Input Method Editor (IME) applications, they:

- [C-1-1] MUST declare the android.software.input_methods feature flag.
- [C-1-2] MUST implement fully Input Management Framework
- [C-1-3] MUST have a preinstalled software keyboard.

Device implementations: * [C-0-1] MUST NOT include a hardware keyboard that does not match one of the formats specified in android.content.res.Configuration.keyboard (QWERTY or 12-key). * SHOULD include additional soft keyboard implementations. * MAY include a hardware keyboard.

7.2.2. Non-touch Navigation

Android includes support for d-pad, trackball, and wheel as mechanisms for non-touch navigation.
Device implementations:

- [C-0-1] MUST report the correct value for android.content.res.Configuration.navigation .

If device implementations lack non-touch navigations, they:

- [C-1-1] MUST provide a reasonable alternative user interface mechanism for the selection and editing of text, compatible with Input Management Engines. The upstream Android open source implementation includes a selection mechanism suitable for use with devices that lack non-touch navigation inputs.

7.2.3. Navigation Keys

The Home , Recents , and Back functions typically provided via an interaction with a dedicated physical button or a distinct portion of the touch screen, are essential to the Android navigation paradigm and therefore, device implementations:

- [C-0-1] MUST provide a user affordance to launch installed applications that have an activity with the <intent-filter> set with ACTION=MAIN and CATEGORY=LAUNCHER or CATEGORY=LEANBACK_LAUNCHER for Television device implementations. The Home function SHOULD be the mechanism for this user affordance.
- SHOULD provide buttons for the Recents and Back function.

If the Home, Recents, or Back functions are provided, they:

- [C-1-1] MUST be accessible with a single action (e.g. tap, double-click or gesture) when any of them are accessible.
- [C-1-2] MUST provide a clear indication of which single action would trigger each function. Having a visible icon imprinted on the button, showing a software icon on the navigation bar portion of the screen, or walking the user through a guided step-by-step demo flow during the out-of-box setup experience are examples of such an indication.

Device implementations:

- [SR] are STRONGLY RECOMMENDED to not provide the input mechanism for the Menu function as it is deprecated in favor of action bar since Android 4.0.

If device implementations provide the Menu function, they:

- [C-2-1] MUST display the action overflow button whenever the action overflow menu popup is not empty and the action bar is visible.
- [C-2-2] MUST NOT modify the position of the action overflow popup displayed by selecting the overflow button in the action bar, but MAY render the action overflow popup at a modified position on the screen when it is displayed by selecting the Menu function.

If device implementations do not provide the Menu function, for backwards compatibility, they: * [C-3-1] MUST make the Menu function available to applications when targetSdkVersion is less than 10, either by a physical button, a software key, or gestures. This Menu function should be accessible unless hidden together with other navigation functions.
If device implementations provide the Assist function , they:

- [C-4-1] MUST make the Assist function accessible with a single action (e.g. tap, double-click or gesture) when other navigation keys are accessible.
- [SR] STRONGLY RECOMMENDED to use long press on HOME function as this designated interaction.

If device implementations use a distinct portion of the screen to display the navigation keys, they:

- [C-5-1] Navigation keys MUST use a distinct portion of the screen, not available to applications, and MUST NOT obscure or otherwise interfere with the portion of the screen available to applications.
- [C-5-2] MUST make available a portion of the display to applications that meets the requirements defined in section 7.1.1 .

- [C-5-3] MUST honor the flags set by the app through the View.setSystemUiVisibility() API method, so that this distinct portion of the screen (a.k.a. the navigation bar) is properly hidden away as documented in the SDK.

If the navigation function is provided as an on-screen, gesture-based action:

- [C-6-1] WindowInsets#getMandatorySystemGestureInsets() MUST only be used to report the Home gesture recognition area.
- [C-6-2] Gestures that start within an exclusion rect as provided by the foreground application via View#setSystemGestureExclusionRects() , but outside of WindowInsets#getMandatorySystemGestureInsets() , MUST NOT be intercepted for the navigation function as long as the exclusion rect is allowed within the max exclusion limit as specified in the documentation for View#setSystemGestureExclusionRects() .
- [C-6-3] MUST send the foreground app a MotionEvent.ACTION_CANCEL event once touches start being intercepted for a system gesture, if the foreground app was previously sent an MotionEvent.ACTION_DOWN event.
- [C-6-4] MUST provide a user affordance to switch to an on-screen, button-based navigation (for example, in Settings).
- SHOULD provide Home function as a swipe up from the bottom edge of the current orientation of the screen.
- SHOULD provide Recents function as a swipe up and hold before release, from the same area as the Home gesture.
- Gestures that start within WindowInsets#getMandatorySystemGestureInsets() SHOULD NOT be affected by exclusion rects provided by the foreground application via View#setSystemGestureExclusionRects() .

If a navigation function is provided from anywhere on the left and right edges of the current orientation of the screen:

- [C-7-1] The navigation function MUST be Back and provided as a swipe from both left and right edges of the current orientation of the screen.
- [C-7-2] If custom swipeable system panels are provided on the left or right edges, they MUST be placed within the top 1/3rd of the screen with a clear, persistent visual indication that dragging in would invoke the aforementioned panels, and hence not Back. A system panel MAY be configured by a user such that it lands below the top 1/3rd of the screen edge(s) but the system panel MUST NOT use longer than 1/3rd of the edge(s).
- [C-7-3] When the foreground app has either the View.SYSTEM_UI_FLAG_IMMERSIVE or View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY flags set, swiping from the edges MUST behave as implemented in AOSP, which is documented in the SDK .
- [C-7-4] When the foreground app has either the View.SYSTEM_UI_FLAG_IMMERSIVE or View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY flags set, custom swipeable system panels MUST be hidden until the user brings in the system bars (a.k.a. navigation and status bar) as implemented in AOSP.

7.2.4. Touchscreen Input

Android includes support for a variety of pointer input systems, such as touchscreens, touch pads, and fake touch input devices. Touchscreen-based device implementations are associated with a display such that the user has the impression of directly manipulating items on screen. Since the user is directly touching the screen, the system does not require any additional affordances to indicate the objects being manipulated.

Device implementations:

- SHOULD have a pointer input system of some kind (either mouse-like or touch).
- SHOULD support fully independently tracked pointers.

If device implementations include a touchscreen (single-touch or better) on a primary Android-compatible display, they:

- [C-1-1] MUST report TOUCHSCREEN_FINGER for the Configuration.touchscreen API field.
- [C-1-2] MUST report the android.hardware.touchscreen and android.hardware.faketouch feature flags.

If device implementations include a touchscreen that can track more than a single touch on a primary Android-compatible display, they:

- [C-2-1] MUST report the appropriate feature flags android.hardware.touchscreen.multitouch , android.hardware.touchscreen.multitouch.distinct , android.hardware.touchscreen.multitouch.jazzhand corresponding to the type of the specific touchscreen on the device.

If device implementations rely on an external input device such as mouse or trackball (i.e. not directly touching the screen) for input on a primary Android-compatible display and meet the fake touch requirements in section 7.2.5 , they:

- [C-3-1] MUST NOT report any feature flag starting with android.hardware.touchscreen .
- [C-3-2] MUST report only android.hardware.faketouch .
- [C-3-3] MUST report TOUCHSCREEN_NOTOUCH for the Configuration.touchscreen API field.

7.2.5. Fake Touch Input

Fake touch interface provides a user input system that approximates a subset of touchscreen capabilities. For example, a mouse or remote control that drives an on-screen cursor approximates touch, but requires the user to first point or focus then click. Numerous input devices like the mouse, trackpad, gyro-based air mouse, gyro-pointer, joystick, and multi-touch trackpad can support fake touch interactions. Android includes the feature constant android.hardware.faketouch, which corresponds to a high-fidelity non-touch (pointer-based) input device such as a mouse or trackpad that can adequately emulate touch-based input (including basic gesture support), and indicates that the device supports an emulated subset of touchscreen functionality.

If device implementations do not include a touchscreen but include another pointer input system which they want to make available, they:

- SHOULD declare support for the android.hardware.faketouch feature flag.

If device implementations declare support for android.hardware.faketouch , they:

- [C-1-1] MUST report the absolute X and Y screen positions of the pointer location and display a visual pointer on the screen.
- [C-1-2] MUST report touch event with the action code that specifies the state change that occurs on the pointer going down or up on the screen .
- [C-1-3] MUST support pointer down and up on an object on the screen, which allows users to emulate tap on an object on the screen.
- [C-1-4] MUST support pointer down, pointer up, pointer down then pointer up in the same place on an object on the screen within a time threshold, which allows users to emulate double tap on an object on the screen.
- [C-1-5] MUST support pointer down on an arbitrary point on the screen, pointer move to any other arbitrary point on the screen, followed by a pointer up, which allows users to emulate a touch drag.
- [C-1-6] MUST support pointer down then allow users to quickly move the object to a different position on the screen and then pointer up on the screen, which allows users to fling an object on the screen.

If device implementations declare support for android.hardware.faketouch.multitouch.distinct , they:

- [C-2-1] MUST declare support for android.hardware.faketouch .
- [C-2-2] MUST support distinct tracking of two or more independent pointer inputs.

If device implementations declare support for android.hardware.faketouch.multitouch.jazzhand , they:

- [C-3-1] MUST declare support for android.hardware.faketouch .
- [C-3-2] MUST support distinct tracking of 5 (tracking a hand of fingers) or more pointer inputs fully independently.

7.2.6. Game Controller Support

7.2.6.1. Button Mappings

Device implementations:

- [C-1-1] MUST be capable to map HID events to the corresponding InputEvent constants as

listed in the below tables. The upstream Android implementation satisfies this requirement.

If device implementations embed a controller or ship with a separate controller in the box that would provide means to input all the events listed in the below tables, they:

- [C-2-1] MUST declare the feature flag android.hardware.gamepad

Button	HID Usage [2]	Android Button
A [1]	0x09 0x0001	KEYCODE_BUTTON_A (96)
B [1]	0x09 0x0002	KEYCODE_BUTTON_B (97)
X [1]	0x09 0x0004	KEYCODE_BUTTON_X (99)
Y [1]	0x09 0x0005	KEYCODE_BUTTON_Y (100)
D-pad up [1] D-pad down [1]	0x01 0x0039 [3]	AXIS_HAT_Y [4]
D-pad left 1 D-pad right [1]	0x01 0x0039 [3]	AXIS_HAT_X [4]
Left shoulder button [1]	0x09 0x0007	KEYCODE_BUTTON_L1 (102)
Right shoulder button [1]	0x09 0x0008	KEYCODE_BUTTON_R1 (103)
Left stick click [1]	0x09 0x000E	KEYCODE_BUTTON_THUMBL (106)
Right stick click [1]	0x09 0x000F	KEYCODE_BUTTON_THUMBR (107)
Home [1]	0x0c 0x0223	KEYCODE_HOME (3)
Back [1]	0x0c 0x0224	KEYCODE_BACK (4)

1 KeyEvent

2 The above HID usages must be declared within a Game pad CA (0x01 0x0005).

3 This usage must have a Logical Minimum of 0, a Logical Maximum of 7, a Physical Minimum of 0, a Physical Maximum of 315, Units in Degrees, and a Report Size of 4. The logical value is defined to be the clockwise rotation away from the vertical axis; for example, a logical value of 0 represents no rotation and the up button being pressed, while a logical value of 1 represents a rotation of 45 degrees and both the up and left keys being pressed.

4 MotionEvent

Analog Controls [1]	HID Usage	Android Button
Left Trigger	0x02 0x00C5	AXIS_LTRIGGER
Right Trigger	0x02 0x00C4	AXIS_RTRIGGER
Left Joystick	0x01 0x0030 0x01 0x0031	AXIS_X AXIS_Y
Right Joystick	0x01 0x0032 0x01 0x0035	AXIS_Z AXIS_RZ

1 MotionEvent

7.2.7. Remote Control

See Section 2.3.1 for device-specific requirements.

7.3. Sensors

If device implementations include a particular sensor type that has a corresponding API for third-party developers, the device implementation MUST implement that API as described in the Android SDK documentation and the Android Open Source documentation on sensors .

Device implementations:

- [C-0-1] MUST accurately report the presence or absence of sensors per the android.content.pm.PackageManager class.

- [C-0-2] MUST return an accurate list of supported sensors via the SensorManager.getSensorList() and similar methods.
- [C-0-3] MUST behave reasonably for all other sensor APIs (for example, by returning true or false as appropriate when applications attempt to register listeners, not calling sensor listeners when the corresponding sensors are not present; etc.).

If device implementations include a particular sensor type that has a corresponding API for third-party developers, they:

- [C-1-1] MUST report all sensor measurements using the relevant International System of Units (metric) values for each sensor type as defined in the Android SDK documentation.
- [C-1-2] MUST report sensor data with a maximum latency of 100 milliseconds + 2 * sample_time for the case of a sensor stream with a maximum requested latency of 0 ms when the application processor is active. This delay does not include any filtering delays.
- [C-1-3] MUST report the first sensor sample within 400 milliseconds + 2 * sample_time of the sensor being activated. It is acceptable for this sample to have an accuracy of 0.
- [C-1-4] For any API indicated by the Android SDK documentation to be a continuous sensor , device implementations MUST continuously provide periodic data samples that SHOULD have a jitter below 3%, where jitter is defined as the standard deviation of the difference of the reported timestamp values between consecutive events.
- [C-1-5] MUST ensure that the sensor event stream MUST NOT prevent the device CPU from entering a suspend state or waking up from a suspend state.
- [C-1-6] MUST report the event time in nanoseconds as defined in the Android SDK documentation, representing the time the event happened and synchronized with the SystemClock.elapsedRealtimeNano() clock.
- [C-SR] Are STRONGLY RECOMMENDED to have timestamp synchronization error below 100 milliseconds, and SHOULD have timestamp synchronization error below 1 millisecond.
- When several sensors are activated, the power consumption SHOULD NOT exceed the sum of the individual sensor's reported power consumption.

The list above is not comprehensive; the documented behavior of the Android SDK and the Android Open Source Documentations on sensors is to be considered authoritative.

If device implementations include a particular sensor type that has a corresponding API for third-party developers, they:

- [C-1-6] MUST set a non-zero resolution for all sensors, and report the value via the Sensor.getResolution() API method.

Some sensor types are composite, meaning they can be derived from data provided by one or more other sensors. (Examples include the orientation sensor and the linear acceleration sensor.)

Device implementations:

- SHOULD implement these sensor types, when they include the prerequisite physical sensors as described in sensor types .

If device implementations include a composite sensor, they:

- [C-2-1] MUST implement the sensor as described in the Android Open Source documentation on composite sensors .

If device implementations include a particular sensor type that has a corresponding API for third-party developers and the sensor only reports one value, then device implementations:

- [C-3-1] MUST set the resolution to 1 for the sensor and report the value via the Sensor.getResolution() API method.

If device implementations include a particular sensor type which supports SensorAdditionalInfo#TYPE_VEC3_CALIBRATION and the sensor is exposed to third-party developers, they:

- [C-4-1] MUST NOT include any fixed, factory-determined calibration parameters in the data provided.

If device implementations include a combination of 3-axis accelerometer, a 3-axis gyroscope sensor, or a magnetometer sensor, they are:

- [C-SR] STRONGLY RECOMMENDED to ensure the accelerometer, gyroscope and magnetometer have a fixed relative position, such that if the device is transformable (e.g. foldable), the sensor axes remain aligned and consistent with the sensor coordinate system throughout all possible device transformation states.

7.3.1. Accelerometer

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to include a 3-axis accelerometer.

If device implementations include a 3-axis accelerometer, they:

- [C-1-1] MUST be able to report events up to a frequency of at least 50 Hz.
- [C-1-2] MUST implement and report TYPE_ACCELEROMETER sensor.
- [C-1-3] MUST comply with the Android sensor coordinate system as detailed in the Android APIs.
- [C-1-4] MUST be capable of measuring from freefall up to four times the gravity(4g) or more on any axis.
- [C-1-5] MUST have a resolution of at least 12-bits.
- [C-1-6] MUST have a standard deviation no greater than 0.05 m/s^, where the standard deviation should be calculated on a per axis basis on samples collected over a period of at least 3 seconds at the fastest sampling rate.
- [SR] are **STRONGLY RECOMMENDED** to implement the TYPE_SIGNIFICANT_MOTION composite sensor.
- [SR] are STRONGLY RECOMMENDED to implement and report [TYPE_ACCELEROMETER_UNCALIBRATED] (https://developer.android.com/reference/android/hardware/Sensor.html#STRING_TYPE_ACCELEROMETER_UNCALIBRATED) sensor. Android devices are STRONGLY RECOMMENDED to meet this requirement so they will be able to upgrade to the future platform release where this might become REQUIRED.
- SHOULD implement the TYPE_SIGNIFICANT_MOTION , TYPE_TILT_DETECTOR , TYPE_STEP_DETECTOR , TYPE_STEP_COUNTER composite sensors as described in the Android SDK document.
- SHOULD report events up to at least 200 Hz.
- SHOULD have a resolution of at least 16-bits.
- SHOULD be calibrated while in use if the characteristics changes over the life cycle and compensated, and preserve the compensation parameters between device reboots.
- SHOULD be temperature compensated.

If device implementations include a 3-axis accelerometer and any of the TYPE_SIGNIFICANT_MOTION , TYPE_TILT_DETECTOR , TYPE_STEP_DETECTOR , TYPE_STEP_COUNTER composite sensors are implemented:

- [C-2-1] The sum of their power consumption MUST always be less than 4 mW.
- SHOULD each be below 2 mW and 0.5 mW for when the device is in a dynamic or static condition.

If device implementations include a 3-axis accelerometer and a 3-axis gyroscope sensor, they:

- [C-3-1] MUST implement the TYPE_GRAVITY and TYPE_LINEAR_ACCELERATION composite sensors.
- [C-SR] Are STRONGLY RECOMMENDED to implement the TYPE_GAME_ROTATION_VECTOR composite sensor.

If device implementations include a 3-axis accelerometer, a 3-axis gyroscope sensor, and a magnetometer sensor, they:

- [C-4-1] MUST implement a TYPE_ROTATION_VECTOR composite sensor.

7.3.2. Magnetometer

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to include a 3-axis magnetometer (compass).

If device implementations include a 3-axis magnetometer, they:

- [C-1-1] MUST implement the TYPE_MAGNETIC_FIELD sensor.
- [C-1-2] MUST be able to report events up to a frequency of at least 10 Hz and SHOULD report events up to at least 50 Hz.
- [C-1-3] MUST comply with the Android sensor coordinate system as detailed in the Android APIs.
- [C-1-4] MUST be capable of measuring between -900 µT and +900 µT on each axis before saturating.
- [C-1-5] MUST have a hard iron offset value less than 700 µT and SHOULD have a value below 200 µT, by placing the magnetometer far from dynamic (current-induced) and static (magnet-induced) magnetic fields.
- [C-1-6] MUST have a resolution equal or denser than 0.6 µT.
- [C-1-7] MUST support online calibration and compensation of the hard iron bias, and preserve the compensation parameters between device reboots.
- [C-1-8] MUST have the soft iron compensation applied—the calibration can be done either while in use or during the production of the device.
- [C-1-9] MUST have a standard deviation, calculated on a per axis basis on samples collected over a period of at least 3 seconds at the fastest sampling rate, no greater than 1.5 µT; SHOULD have a standard deviation no greater than 0.5 µT.
- [C-SR] Are STRONGLY RECOMMENDED to implement TYPE_MAGNETIC_FIELD_UNCALIBRATED sensor.

If device implementations include a 3-axis magnetometer, an accelerometer sensor, and a 3-axis gyroscope sensor, they:

- [C-2-1] MUST implement a TYPE_ROTATION_VECTOR composite sensor.

If device implementations include a 3-axis magnetometer, an accelerometer, they:

- MAY implement the TYPE_GEOMAGNETIC_ROTATION_VECTOR sensor.

If device implementations include a 3-axis magnetometer, an accelerometer and TYPE_GEOMAGNETIC_ROTATION_VECTOR sensor, they:

- [C-3-1] MUST consume less than 10 mW.
- SHOULD consume less than 3 mW when the sensor is registered for batch mode at 10 Hz.

7.3.3. GPS

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to include a GPS/GNSS receiver.

If device implementations include a GPS/GNSS receiver and report the capability to applications through the android.hardware.location.gps feature flag, they:

- [C-1-1] MUST support location outputs at a rate of at least 1 Hz when requested via LocationManager#requestLocationUpdate .
- [C-1-2] MUST be able to determine the location in open-sky conditions (strong signals, negligible multipath, HDOP < 2) within 10 seconds (fast time to first fix), when connected to a 0.5 Mbps or faster data speed internet connection. This requirement is typically met by the use of some form of Assisted or Predicted GPS/GNSS technique to minimize GPS/GNSS lock-on time (Assistance data includes Reference Time, Reference Location and Satellite Ephemeris/Clock).
  - [C-1-6] After making such a location calculation, device implementations MUST determine its location, in open sky, within 5 seconds, when location requests are restarted, up to an hour after the initial location calculation, even when the subsequent request is made without a data connection, and/or after a power cycle.
- In open sky conditions after determining the location, while stationary or moving with less than 1 meter per second squared of acceleration:
  - [C-1-3] MUST be able to determine location within 20 meters, and speed within 0.5 meters per second, at least 95% of the time.
  - [C-1-4] MUST simultaneously track and report via GnssStatus.Callback at least 8

satellites from one constellation.
    - SHOULD be able to simultaneously track at least 24 satellites, from multiple constellations (e.g. GPS + at least one of Glonass, Beidou, Galileo).
    - [C-SR] Are STRONGLY RECOMMENDED to continue to deliver normal GPS/GNSS location outputs through GNSS Location Provider API's during an emergency phone call.
    - [C-SR] Are STRONGLY RECOMMENDED to report GNSS measurements from all constellations tracked (as reported in GnssStatus messages), with the exception of SBAS.
    - [C-SR] Are STRONGLY RECOMMENDED to report AGC, and Frequency of GNSS measurement.
    - [C-SR] Are STRONGLY RECOMMENDED to report all accuracy estimates (including Bearing, Speed, and Vertical) as part of each GPS/GNSS location.
    - [C-SR] Are STRONGLY RECOMMENDED to report GNSS measurements, as soon as they are found, even if a location calculated from GPS/GNSS is not yet reported.
    - [C-SR] Are STRONGLY RECOMMENDED to report GNSS pseudoranges and pseudorange rates, that, in open-sky conditions after determining the location, while stationary or moving with less than 0.2 meter per second squared of acceleration, are sufficient to calculate position within 20 meters, and speed within 0.2 meters per second, at least 95% of the time.

7.3.4. Gyroscope

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to include a gyroscope sensor.

If device implementations include a 3-axis gyroscope, they:

- [C-1-1] MUST be able to report events up to a frequency of at least 50 Hz.
- [C-1-2] MUST implement the TYPE_GYROSCOPE sensor and are STRONGLY RECOMMENDED to also implement the TYPE_GYROSCOPE_UNCALIBRATED sensor.
- [C-1-4] MUST have a resolution of 12-bits or more and SHOULD have a resolution of 16-bits or more.
- [C-1-5] MUST be temperature compensated.
- [C-1-6] MUST be calibrated and compensated while in use, and preserve the compensation parameters between device reboots.
- [C-1-7] MUST have a variance no greater than 1e-7 rad^2 / s^2 per Hz (variance per Hz, or rad^2 / s). The variance is allowed to vary with the sampling rate, but MUST be constrained by this value. In other words, if you measure the variance of the gyro at 1 Hz sampling rate it SHOULD be no greater than 1e-7 rad^2/s^2.
- [SR] Calibration error is STRONGLY RECOMMENDED to be less than 0.01 rad/s when device is stationary at room temperature.
- SHOULD report events up to at least 200 Hz.

If device implementations include a 3-axis gyroscope, an accelerometer sensor and a magnetometer sensor, they:

- [C-2-1] MUST implement a TYPE_ROTATION_VECTOR composite sensor.

If device implementations include a 3-axis accelerometer and a 3-axis gyroscope sensor, they:

- [C-3-1] MUST implement the TYPE_GRAVITY and TYPE_LINEAR_ACCELERATION composite sensors.
- [C-SR] Are STRONGLY RECOMMENDED to implement the TYPE_GAME_ROTATION_VECTOR composite sensor.

7.3.5. Barometer

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to include a barometer (ambient air pressure sensor).

If device implementations include a barometer, they:

- [C-1-1] MUST implement and report TYPE_PRESSURE sensor.
- [C-1-2] MUST be able to deliver events at 5 Hz or greater.
- [C-1-3] MUST be temperature compensated.
- [SR] STRONGLY RECOMMENDED to be able to report pressure measurements in the range 300hPa to 1100hPa.
- SHOULD have an absolute accuracy of 1hPa.
- SHOULD have a relative accuracy of 0.12hPa over 20hPa range (equivalent to ~1m accuracy over ~200m change at sea level).

7.3.6. Thermometer

If device implementations include an ambient thermometer (temperature sensor), they:

- [C-1-1] MUST define SENSOR_TYPE_AMBIENT_TEMPERATURE for the ambient temperature sensor and the sensor MUST measure the ambient (room/vehicle cabin) temperature from where the user is interacting with the device in degrees Celsius.

If device implementations include a thermometer sensor that measures a temperature other than ambient temperature, such as CPU temperature, they:

- [C-2-1] MUST NOT define SENSOR_TYPE_AMBIENT_TEMPERATURE for the temperature sensor.

7.3.7. Photometer

- Device implementations MAY include a photometer (ambient light sensor).

7.3.8. Proximity Sensor

- Device implementations MAY include a proximity sensor.

If device implementations include a proximity sensor, they:

- [C-1-1] MUST measure the proximity of an object in the same direction as the screen. That is, the proximity sensor MUST be oriented to detect objects close to the screen, as the primary intent of this sensor type is to detect a phone in use by the user. If device implementations include a proximity sensor with any other orientation, it MUST NOT be accessible through this API.
- [C-1-2] MUST have 1-bit of accuracy or more.

7.3.9. High Fidelity Sensors

If device implementations include a set of higher quality sensors as defined in this section, and make available them to third-party apps, they:

- [C-1-1] MUST identify the capability through the android.hardware.sensor.hifi_sensors feature flag.

If device implementations declare android.hardware.sensor.hifi_sensors , they:

- [C-2-1] MUST have a TYPE_ACCELEROMETER sensor which:
  - MUST have a measurement range between at least -8g and +8g, and is STRONGLY RECOMMENDED to have a measurement range between at least -16g and +16g.
  - MUST have a measurement resolution of at least 2048 LSB/g.
  - MUST have a minimum measurement frequency of 12.5 Hz or lower.
  - MUST have a maximum measurement frequency of 400 Hz or higher; SHOULD support the SensorDirectChannel RATE_VERY_FAST .
  - MUST have a measurement noise not above 400 µg/√Hz.
  - MUST implement a non-wake-up form of this sensor with a buffering capability of at least 3000 sensor events.
  - MUST have a batching power consumption not worse than 3 mW.

    - [C-SR] Is STRONGLY RECOMMENDED to have 3dB measurement bandwidth of at least 80% of Nyquist frequency, and white noise spectrum within this bandwidth.
    - SHOULD have an acceleration random walk less than 30 µg √Hz tested at room temperature.
    - SHOULD have a bias change vs. temperature of ≤ +/- 1 mg/°C.
    - SHOULD have a best-fit line non-linearity of ≤ 0.5%, and sensitivity change vs. temperature of ≤ 0.03%/C°.
    - SHOULD have cross-axis sensitivity of < 2.5 % and variation of cross-axis sensitivity < 0.2% in device operation temperature range.
- [C-2-2] MUST have a TYPE_ACCELEROMETER_UNCALIBRATED with the same quality requirements as TYPE_ACCELEROMETER .
- [C-2-3] MUST have a TYPE_GYROSCOPE sensor which:
    - MUST have a measurement range between at least -1000 and +1000 dps.
    - MUST have a measurement resolution of at least 16 LSB/dps.
    - MUST have a minimum measurement frequency of 12.5 Hz or lower.
    - MUST have a maximum measurement frequency of 400 Hz or higher; SHOULD support the SensorDirectChannel RATE_VERY_FAST .
    - MUST have a measurement noise not above 0.014°/s/√Hz.
    - [C-SR] Is STRONGLY RECOMMENDED to have 3dB measurement bandwidth of at least 80% of Nyquist frequency, and white noise spectrum within this bandwidth.
    - SHOULD have a rate random walk less than 0.001 °/s √Hz tested at room temperature.
    - SHOULD have a bias change vs. temperature of ≤ +/- 0.05 °/ s / °C.
    - SHOULD have a sensitivity change vs. temperature of ≤ 0.02% / °C.
    - SHOULD have a best-fit line non-linearity of ≤ 0.2%.
    - SHOULD have a noise density of ≤ 0.007 °/s/√Hz.
    - SHOULD have calibration error less than 0.002 rad/s in temperature range 10 ~ 40 ℃ when device is stationary.
    - SHOULD have g-sensitivity less than 0.1°/s/g.
    - SHOULD have cross-axis sensitivity of < 4.0 % and cross-axis sensitivity variation < 0.3% in device operation temperature range.
- [C-2-4] MUST have a TYPE_GYROSCOPE_UNCALIBRATED with the same quality requirements as TYPE_GYROSCOPE .
- [C-2-5] MUST have a TYPE_GEOMAGNETIC_FIELD sensor which:
    - MUST have a measurement range between at least -900 and +900 µT.
    - MUST have a measurement resolution of at least 5 LSB/uT.
    - MUST have a minimum measurement frequency of 5 Hz or lower.
    - MUST have a maximum measurement frequency of 50 Hz or higher.
    - MUST have a measurement noise not above 0.5 uT.
- [C-2-6] MUST have a TYPE_MAGNETIC_FIELD_UNCALIBRATED with the same quality requirements as TYPE_GEOMAGNETIC_FIELD and in addition:
    - MUST implement a non-wake-up form of this sensor with a buffering capability of at least 600 sensor events.
    - [C-SR] Is STRONGLY RECOMMENDED to have white noise spectrum from 1 Hz to at least 10 Hz when the report rate is 50 Hz or higher.
- [C-2-7] MUST have a TYPE_PRESSURE sensor which:
    - MUST have a measurement range between at least 300 and 1100 hPa.
    - MUST have a measurement resolution of at least 80 LSB/hPa.
    - MUST have a minimum measurement frequency of 1 Hz or lower.
    - MUST have a maximum measurement frequency of 10 Hz or higher.
    - MUST have a measurement noise not above 2 Pa/√Hz.
    - MUST implement a non-wake-up form of this sensor with a buffering capability of at least 300 sensor events.
    - MUST have a batching power consumption not worse than 2 mW.
- [C-2-8] MUST have a TYPE_GAME_ROTATION_VECTOR sensor.
- [C-2-9] MUST have a TYPE_SIGNIFICANT_MOTION sensor which:
    - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving.

- [C-2-10] MUST have a TYPE_STEP_DETECTOR sensor which:
  - MUST implement a non-wake-up form of this sensor with a buffering capability of at least 100 sensor events.
  - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving.
  - MUST have a batching power consumption not worse than 4 mW.
- [C-2-11] MUST have a TYPE_STEP_COUNTER sensor which:
  - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving.
- [C-2-12] MUST have a TILT_DETECTOR sensor which:
  - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving.
- [C-2-13] The event timestamp of the same physical event reported by the Accelerometer, Gyroscope, and Magnetometer MUST be within 2.5 milliseconds of each other. The event timestamp of the same physical event reported by the Accelerometer and Gyroscope SHOULD be within 0.25 milliseconds of each other.
- [C-2-14] MUST have Gyroscope sensor event timestamps on the same time base as the camera subsystem and within 1 milliseconds of error.
- [C-2-15] MUST deliver samples to applications within 5 milliseconds from the time when the data is available on any of the above physical sensors to the application.
- [C-2-16] MUST NOT have a power consumption higher than 0.5 mW when device is static and 2.0 mW when device is moving when any combination of the following sensors are enabled:
  - SENSOR_TYPE_SIGNIFICANT_MOTION
  - SENSOR_TYPE_STEP_DETECTOR
  - SENSOR_TYPE_STEP_COUNTER
  - SENSOR_TILT_DETECTORS
- [C-2-17] MAY have a TYPE_PROXIMITY sensor, but if present MUST have a minimum buffer capability of 100 sensor events.

Note that all power consumption requirements in this section do not include the power consumption of the Application Processor. It is inclusive of the power drawn by the entire sensor chain—the sensor, any supporting circuitry, any dedicated sensor processing system, etc.

If device implementations include direct sensor support, they:

- [C-3-1] MUST correctly declare support of direct channel types and direct report rates level through the isDirectChannelTypeSupported and getHighestDirectReportRateLevel API.
- [C-3-2] MUST support at least one of the two sensor direct channel types for all sensors that declare support for sensor direct channel.
  - TYPE_HARDWARE_BUFFER
  - TYPE_MEMORY_FILE
- SHOULD support event reporting through sensor direct channel for primary sensor (non-wakeup variant) of the following types:
  - TYPE_ACCELEROMETER
  - TYPE_ACCELEROMETER_UNCALIBRATED
  - TYPE_GYROSCOPE
  - TYPE_GYROSCOPE_UNCALIBRATED
  - TYPE_MAGNETIC_FIELD
  - TYPE_MAGNETIC_FIELD_UNCALIBRATED

7.3.10. Biometric Sensors

For additional background on Measuring Biometric Unlock Security, please see Measuring Biometric Security documentation .

If device implementations include a secure lock screen, they:

- SHOULD include a biometric sensor

Biometric sensors can be classified as **Class 3** (formerly **Strong**), **Class 2** (formerly **Weak**), or **Class 1** (formerly **Convenience**) based on their spoof and imposter acceptance rates, and on the security of the biometric pipeline. This classification determines the capabilities the biometric sensor has to interface with the platform and with third-party applications. Sensors are classified as **Class 1** by default, and need to meet additional requirements as detailed below if they wish to be classified as

either **Class 2** or **Class 3** . Both **Class 2** and **Class 3** biometrics get additional capabilities as detailed below.

If device implementations make a biometric sensor available to third-party applications via android.hardware.biometrics.BiometricManager , android.hardware.biometrics.BiometricPrompt , and android.provider.Settings.ACTION_BIOMETRIC_ENROLL , they:

- [C-4-1] MUST meet the requirements for **Class 3** or **Class 2** biometric as defined in this document.
- [C-4-2] MUST recognize and honor each parameter name defined as a constant in the Authenticators class and any combinations thereof. Conversely, MUST NOT honor or recognize integer constants passed to the canAuthenticate(int) and setAllowedAuthenticators(int) methods other than those documented as public constants in Authenticators and any combinations thereof.
- [C-4-3] MUST implement the ACTION_BIOMETRIC_ENROLL action on devices that have either **Class 3** or **Class 2** biometrics. This action MUST only present the enrollment entry points for **Class 3** or **Class 2** biometrics.

If device implementations support passive biometrics, they:

- [C-5-1] MUST by default require an additional confirmation step (e.g. a button press).
- [C-SR] Are STRONGLY RECOMMENDED to have a setting to allow users to override application preference and always require accompanying confirmation step.
- [C-SR] Are STRONGLY RECOMMENDED to have the confirm action be secured such that an operating system or kernel compromise cannot spoof it. For example, this means that the confirm action based on a physical button is routed through an input-only general-purpose input/output (GPIO) pin of a secure element (SE) that cannot be driven by any other means than a physical button press.
- [C-5-2] MUST additionally implement an implicit authentication flow (without confirmation step) corresponding to setConfirmationRequired(boolean) , which applications can set to utilize for sign-in flows.

If device implementations have multiple biometric sensors, they:

- [C-SR] Are STRONGLY RECOMMENDED to require only one biometric be confirmed per authentication (e.g. if both fingerprint and face sensors are available on the device, onAuthenticationSucceeded should be sent after any one of them is confirmed).

In order for device implementations to allow access to keystore keys to third-party applications, they:

- [C-6-1] MUST meet the requirements for **Class 3** as defined in this section below.
- [C-6-2] MUST present only **Class 3** biometrics when the authentication requires BIOMETRIC_STRONG , or the authentication is invoked with a CryptoObject .

If device implementations wish to treat a biometric sensor as **Class 1** (formerly **Convenience**), they:

- [C-1-1] MUST have a false acceptance rate less than 0.002%.
- [C-1-2] MUST disclose that this mode may be less secure than a strong PIN, pattern, or password and clearly enumerate the risks of enabling it, if the spoof and imposter acceptance rates are higher than 7% as measured by the Android Biometrics Test Protocols .
- [C-1-3] MUST rate limit attempts for at least 30 seconds after five false trials for biometric verification - where a false trial is one with an adequate capture quality (BIOMETRIC_ACQUIRED_GOOD) that does not match an enrolled biometric.
- [C-1-4] MUST prevent adding new biometrics without first establishing a chain of trust by having the user confirm existing or add a new device credential (PIN/pattern/password) that's secured by TEE; the Android Open Source Project implementation provides the mechanism in the framework to do so.
- [C-1-5] MUST completely remove all identifiable biometric data for a user when the user's account is removed (including via a factory reset).
- [C-1-6] MUST honor the individual flag for that biometric (i.e. DevicePolicyManager.KEYGUARD_DISABLE_FINGERPRINT , DevicePolicymanager.KEYGUARD_DISABLE_FACE , or DevicePolicymanager.KEYGUARD_DISABLE_IRIS).
- [C-1-7] MUST challenge the user for the recommended primary authentication (e.g. PIN, pattern, password) once every 24 hours or less for new devices launching with Android version 10, once every 72 hours or less for devices upgrading from earlier Android

version.

- [C-1-8] MUST challenge the user for the recommended primary authentication (eg: PIN, pattern, password) after one of the following:
  - a 4-hour idle timeout period, OR
  - 3 failed biometric authentication attempts.
  - The idle timeout period and the failed authentication count is reset after any successful confirmation of the device credentials.
- [C-SR] Are STRONGLY RECOMMENDED to use the logic in the framework provided by the Android Open Source Project to enforce constraints specified in [C-1-7] and [C-1-8] for new devices.
- [C-SR] Are STRONGLY RECOMMENDED to have a false rejection rate of less than 10%, as measured on the device.
- [C-SR] Are STRONGLY RECOMMENDED to have a latency below 1 second, measured from when the biometric is detected, until the screen is unlocked, for each enrolled biometric.

If device implementations wish to treat a biometric sensor as **Class 2** (formerly **Weak**), they:

- [C-2-1] MUST meet all requirements for **Class 1** above.
- [C-2-2] MUST have a spoof and imposter acceptance rate not higher than 20% as measured by the Android Biometrics Test Protocols .
- [C-2-3] MUST perform the biometric matching in an isolated execution environment outside Android user or kernel space, such as the Trusted Execution Environment (TEE), or on a chip with a secure channel to the isolated execution environment.
- [C-2-4] MUST have all identifiable data encrypted and cryptographically authenticated such that they cannot be acquired, read or altered outside of the isolated execution environment or a chip with a secure channel to the isolated execution environment as documented in the implementation guidelines on the Android Open Source Project site.
- [C-2-5] For camera based biometrics, while biometric based authentication or enrollment is happening:
  - MUST operate the camera in a mode that prevents camera frames from being read or altered outside the isolated execution environment or a chip with a secure channel to the isolated execution environment.
  - For RGB single-camera solutions, the camera frames CAN be readable outside the isolated execution environment to support operations such as preview for enrollment, but MUST still NOT be alterable.
- [C-2-6] MUST NOT enable third-party applications to distinguish between individual biometric enrollments.
- [C-2-7] MUST NOT allow unencrypted access to identifiable biometric data or any data derived from it (such as embeddings) to the Application Processor outside the context of the TEE.
- [C-2-8] MUST have a secure processing pipeline such that an operating system or kernel compromise cannot allow data to be directly injected to falsely authenticate as the user.

  If device implementations are already launched on an earlier Android version and cannot meet the requirement C-2-8 through a system software update, they MAY be exempted from the requirement.
- [C-SR] Are STRONGLY RECOMMENDED to include liveness detection for all biometric modalities and attention detection for Face biometrics.

If device implementations wish to treat a biometric sensor as **Class 3** (formerly **Strong**), they:

- [C-3-1] MUST meet all the requirements of **Class 2** above, except for [C-1-7] and [C-1-8]. Upgrading devices from an earlier Android version are not exempted from C-2-7.
- [C-3-2] MUST have a hardware-backed keystore implementation.
- [C-3-3] MUST have a spoof and imposter acceptance rate not higher than 7% as measured by the Android Biometrics Test Protocols .
- [C-3-4] MUST challenge the user for the recommended primary authentication (e.g. PIN, pattern, password) once every 72 hours or less.

7.3.12. Pose Sensor

Device implementations:

- MAY support pose sensor with 6 degrees of freedom.

If device implementations support pose sensor with 6 degrees of freedom, they:

- [C-1-1] MUST implement and report TYPE_POSE_6DOF sensor.
- [C-1-2] MUST be more accurate than the rotation vector alone.

7.3.13. Hinge Angle Sensor

If device implementations support a hinge angle sensor, they:

- [C-1-1] MUST implement and report TYPE_HINGLE_ANGLE .
- [C-1-2] MUST support at least two readings between 0 and 360 degrees (inclusive i.e including 0 and 360 degrees).
- [C-1-3] MUST return a wakeup sensor for getDefaultSensor(SENSOR_TYPE_HINGE_ANGLE) .

7.4. Data Connectivity

7.4.1. Telephony

“Telephony” as used by the Android APIs and this document refers specifically to hardware related to placing voice calls and sending SMS messages via a GSM or CDMA network. While these voice calls may or may not be packet-switched, they are for the purposes of Android considered independent of any data connectivity that may be implemented using the same network. In other words, the Android “telephony” functionality and APIs refer specifically to voice calls and SMS. For instance, device implementations that cannot place calls or send/receive SMS messages are not considered a telephony device, regardless of whether they use a cellular network for data connectivity.

- Android MAY be used on devices that do not include telephony hardware. That is, Android is compatible with devices that are not phones.

If device implementations include GSM or CDMA telephony, they:

- [C-1-1] MUST declare the android.hardware.telephony feature flag and other sub-feature flags according to the technology.
- [C-1-2] MUST implement full support for the API for that technology.

If device implementations do not include telephony hardware, they:

- [C-2-1] MUST implement the full APIs as no-ops.

If device implementations support eUICCs or eSIMs/embedded SIMs and include a proprietary mechanism to make eSIM functionality available for third-party developers, they:

- [C-3-1] MUST provide a complete implementation of the EuiccManager API .

7.4.1.1. Number Blocking Compatibility

If device implementations report the android.hardware.telephony feature , they:

- [C-1-1] MUST include number blocking support
- [C-1-2] MUST fully implement BlockedNumberContract and the corresponding API as described in the SDK documentation.
- [C-1-3] MUST block all calls and messages from a phone number in 'BlockedNumberProvider' without any interaction with apps. The only exception is when number blocking is temporarily lifted as described in the SDK documentation.
- [C-1-4] MUST NOT write to the platform call log provider for a blocked call.
- [C-1-5] MUST NOT write to the Telephony provider for a blocked message.
- [C-1-6] MUST implement a blocked numbers management UI, which is opened with the intent returned by TelecomManager.createManageBlockedNumbersIntent() method.
- [C-1-7] MUST NOT allow secondary users to view or edit the blocked numbers on the device as the Android platform assumes the primary user to have full control of the telephony services, a single instance, on the device. All blocking related UI MUST be hidden for secondary users and the blocked list MUST still be respected.
- SHOULD migrate the blocked numbers into the provider when a device updates to Android 7.0.

7.4.1.2. Telecom API

If device implementations report android.hardware.telephony , they:

- [C-1-1] MUST support the ConnectionService APIs described in the SDK .
- [C-1-2] MUST display a new incoming call and provide user affordance to accept or reject the incoming call when the user is on an ongoing call that is made by a third-party app that does not support the hold feature specified via CAPABILITY_SUPPORT_HOLD .
- [C-1-3] MUST have an application that implements InCallService .
- [C-SR] Are STRONGLY RECOMMENDED to notify the user that answering an incoming call will drop an ongoing call.

  The AOSP implementation meets these requirements by a heads-up notification which indicates to the user that answering an incoming call will cause the other call to be dropped.
- [C-SR] Are STRONGLY RECOMMENDED to preload the default dialer app that shows a call log entry and the name of a third-party app in its call log when the third-party app sets the EXTRA_LOG_SELF_MANAGED_CALLS extras key on its PhoneAccount to true .
- [C-SR] Are STRONGLY RECOMMENDED to handle the audio headset's KEYCODE_MEDIA_PLAY_PAUSE and KEYCODE_HEADSETHOOK events for the android.telecom APIs as below:
  - Call Connection.onDisconnect() when a short press of the key event is detected during an ongoing call.
  - Call Connection.onAnswer() when a short press of the key event is detected during an incoming call.
  - Call Connection.onReject() when a long press of the key event is detected during an incoming call.
  - Toggle the mute status of the CallAudioState .

7.4.2. IEEE 802.11 (Wi-Fi)

Device implementations:

- SHOULD include support for one or more forms of 802.11.

If device implementations include support for 802.11 and expose the functionality to a third-party application, they:

- [C-1-1] MUST implement the corresponding Android API.
- [C-1-2] MUST report the hardware feature flag android.hardware.wifi .
- [C-1-3] MUST implement the multicast API as described in the SDK documentation.
- [C-1-4] MUST support multicast DNS (mDNS) and MUST NOT filter mDNS packets (224.0.0.251) at any time of operation including:
  - Even when the screen is not in an active state.
  - For Android Television device implementations, even when in standby power states.
- [C-1-5] MUST NOT treat the WifiManager.enableNetwork() API method call as a sufficient indication to switch the currently active Network that is used by default for application traffic and is returned by ConnectivityManager API methods such as getActiveNetwork and registerDefaultNetworkCallback . In other words, they MAY only disable the Internet access provided by any other network provider (e.g. mobile data) if they successfully validate that the Wi-Fi network is providing Internet access.
- [C-1-6] Are STRONGLY RECOMMENDED to, when the ConnectivityManager.reportNetworkConnectivity() API method is called, re-evaluate the Internet access on the Network and, once the evaluation determines that the current Network no longer provides Internet access, switch to any other available network (e.g. mobile data) that provides Internet access.
- [C-SR] Are STRONGLY RECOMMENDED to randomize the source MAC address and sequence number of probe request frames, once at the beginning of each scan, while STA is disconnected.
  - Each group of probe request frames comprising one scan should use one consistent MAC address (SHOULD NOT randomize MAC address halfway through a scan).
  - Probe request sequence number should iterate as normal (sequentially) between the probe requests in a scan.

    - Probe request sequence number should randomize between the last probe request of a scan and the first probe request of the next scan.
- [C-SR] Are STRONGLY RECOMMENDED, while STA is disconnected, to allow only the following elements in probe request frames:
    - SSID Parameter Set (0)
    - DS Parameter Set (3)

If device implementations include support for Wi-Fi power save mode as defined in IEEE 802.11 standard, they:

- [C-3-1] MUST turn off Wi-Fi power save mode whenever an app acquires WIFI_MODE_FULL_HIGH_PERF lock or WIFI_MODE_FULL_LOW_LATENCY lock via WifiManager.createWifiLock() and WifiManager.WifiLock.acquire() APIs and the lock is active.
- [C-3-2] The average round trip latency between the device and an access point while the device is in a Wi-Fi Low Latency Lock (WIFI_MODE_FULL_LOW_LATENCY) mode MUST be smaller than the latency during a Wi-Fi High Perf Lock (WIFI_MODE_FULL_HIGH_PERF) mode.
- [C-SR] Are STRONGLY RECOMMENDED to minimize Wi-Fi round trip latency whenever a Low Latency Lock (WIFI_MODE_FULL_LOW_LATENCY) is acquired and takes effect.

If device implementations support Wi-Fi and use Wi-Fi for location scanning, they:

- [C-2-1] MUST provide a user affordance to enable/disable the value read through the WifiManager.isScanAlwaysAvailable API method.

7.4.2.1. Wi-Fi Direct

Device implementations:

- SHOULD include support for Wi-Fi Direct (Wi-Fi peer-to-peer).

If device implementations include support for Wi-Fi Direct, they:

- [C-1-1] MUST implement the corresponding Android API as described in the SDK documentation.
- [C-1-2] MUST report the hardware feature android.hardware.wifi.direct .
- [C-1-3] MUST support regular Wi-Fi operation.
- [C-1-4] MUST support Wi-Fi and Wi-Fi Direct operations concurrently.

7.4.2.2. Wi-Fi Tunneled Direct Link Setup

Device implementations:

- SHOULD include support for Wi-Fi Tunneled Direct Link Setup (TDLS) as described in the Android SDK Documentation.

If device implementations include support for TDLS and TDLS is enabled by the WiFiManager API, they:

- [C-1-1] MUST declare support for TDLS through [WifiManager.isTdlsSupported] (https://developer.android.com/reference/android/net/wifi/WifiManager.html#isTdlsSupported%28%29).
- SHOULD use TDLS only when it is possible AND beneficial.
- SHOULD have some heuristic and NOT use TDLS when its performance might be worse than going through the Wi-Fi access point.

7.4.2.3. Wi-Fi Aware

Device implementations:

- SHOULD include support for Wi-Fi Aware .

If device implementations include support for Wi-Fi Aware and expose the functionality to third-party apps, then they:

- [C-1-1] MUST implement the WifiAwareManager APIs as described in the SDK documentation .
- [C-1-2] MUST declare the android.hardware.wifi.aware feature flag.
- [C-1-3] MUST support Wi-Fi and Wi-Fi Aware operations concurrently.
- [C-1-4] MUST randomize the Wi-Fi Aware management interface address at intervals no longer than 30 minutes and whenever Wi-Fi Aware is enabled.

If device implementations include support for Wi-Fi Aware and Wi-Fi Location as described in Section 7.4.2.5 and exposes these functionalities to third-party apps, then they:

- [C-2-1] MUST implement the location-aware discovery APIs: setRangingEnabled , setMinDistanceMm , setMaxDistanceMm , and onServiceDiscoveredWithinRange .

7.4.2.4. Wi-Fi Passpoint

Device implementations:

- SHOULD include support for Wi-Fi Passpoint .

If device implementations include support for Wi-Fi Passpoint, they:

- [C-1-1] MUST implement the Passpoint related WifiManager APIs as described in the SDK documentation .
- [C-1-2] MUST support IEEE 802.11u standard, specifically related to Network Discovery and Selection, such as Generic Advertisement Service (GAS) and Access Network Query Protocol (ANQP).

Conversely if device implementations do not include support for Wi-Fi Passpoint:

- [C-2-1] The implementation of the Passpoint related WifiManager APIs MUST throw an UnsupportedOperationException .

7.4.2.5. Wi-Fi Location (Wi-Fi Round Trip Time - RTT)

Device implementations:

- SHOULD include support for Wi-Fi Location .

If device implementations include support for Wi-Fi Location and expose the functionality to third-party apps, then they:

- [C-1-1] MUST implement the WifiRttManager APIs as described in the SDK documentation .
- [C-1-2] MUST declare the android.hardware.wifi.rtt feature flag.
- [C-1-3] MUST randomize the source MAC address for each RTT burst which is executed while the Wi-Fi interface on which the RTT is being executed is not associated to an Access Point.

7.4.2.6. Wi-Fi Keepalive Offload

Device implementations:

- SHOULD include support for Wi-Fi keepalive offload.

If device implementations include support for Wi-Fi keepalive offload and expose the functionality to third-party apps, they:

- [C-1-1] MUST support the SocketKeepAlive API.
- [C-1-2] MUST support at least three concurrent keepalive slots over Wi-Fi and at least one keepalive slot over cellular.

If device implementations do not include support for Wi-Fi keepalive offload, they:

- [C-2-1] MUST return ERROR_UNSUPPORTED .

7.4.2.7. Wi-Fi Easy Connect (Device Provisioning Protocol)

Device implementations:

- SHOULD include support for Wi-Fi Easy Connect (DPP) .

If device implementations include support for Wi-Fi Easy Connect and expose the functionality to third-party apps, they:

- [C-1-1] MUST have the WifiManager#isEasyConnectSupported() method return true .

7.4.3. Bluetooth

If device implementations support Bluetooth Audio profile, they:

- SHOULD support Advanced Audio Codecs and Bluetooth Audio Codecs (e.g. LDAC).

If device implementations support HFP, A2DP and AVRCP, they:

- SHOULD support at least 5 total connected devices.

If device implementations declare android.hardware.vr.high_performance feature, they:

- [C-1-1] MUST support Bluetooth 4.2 and Bluetooth LE Data Length Extension.

Android includes support for Bluetooth and Bluetooth Low Energy .
If device implementations include support for Bluetooth and Bluetooth Low Energy, they:

- [C-2-1] MUST declare the relevant platform features (android.hardware.bluetooth and android.hardware.bluetooth_le respectively) and implement the platform APIs.
- SHOULD implement relevant Bluetooth profiles such as A2DP, AVRCP, OBEX, HFP, etc. as appropriate for the device.

If device implementations include support for Bluetooth Low Energy, they:

- [C-3-1] MUST declare the hardware feature android.hardware.bluetooth_le .
- [C-3-2] MUST enable the GATT (generic attribute profile) based Bluetooth APIs as described in the SDK documentation and android.bluetooth .
- [C-3-3] MUST report the correct value for BluetoothAdapter.isOffloadedFilteringSupported() to indicate whether the filtering logic for the ScanFilter API classes is implemented.
- [C-3-4] MUST report the correct value for BluetoothAdapter.isMultipleAdvertisementSupported() to indicate whether Low Energy Advertising is supported.
- [C-3-5] MUST implement a Resolvable Private Address (RPA) timeout no longer than 15 minutes and rotate the address at timeout to protect user privacy. To prevent timing attacks, timeout intervals MUST also be randomized between 5 and 15 minutes.
- SHOULD support offloading of the filtering logic to the bluetooth chipset when implementing the ScanFilter API .
- SHOULD support offloading of the batched scanning to the bluetooth chipset.
- SHOULD support multi advertisement with at least 4 slots.

If device implementations support Bluetooth LE and use Bluetooth LE for location scanning, they:

- [C-4-1] MUST provide a user affordance to enable/disable the value read through the System API BluetoothAdapter.isBleScanAlwaysAvailable() .

If device implementations include support for Bluetooth LE and Hearing Aids Profile, as described in Hearing Aid Audio Support Using Bluetooth LE , they:

- [C-5-1] MUST return true for BluetoothAdapter.getProfileProxy(context, listener, BluetoothProfile.HEARING_AID) .

7.4.4. Near-Field Communications

Device implementations:

- SHOULD include a transceiver and related hardware for Near-Field Communications (NFC).
- [C-0-1] MUST implement android.nfc.NdefMessage and android.nfc.NdefRecord APIs even if they do not include support for NFC or declare the android.hardware.nfc feature as the classes represent a protocol-independent data representation format.

If device implementations include NFC hardware and plan to make it available to third-party apps, they:

- [C-1-1] MUST report the android.hardware.nfc feature from the android.content.pm.PackageManager.hasSystemFeature() method .
- MUST be capable of reading and writing NDEF messages via the following NFC standards as below:
- [C-1-2] MUST be capable of acting as an NFC Forum reader/writer (as defined by the NFC Forum technical specification NFCForum-TS-DigitalProtocol-1.0) via the following NFC standards:
  - NfcA (ISO14443-3A)
  - NfcB (ISO14443-3B)
  - NfcF (JIS X 6319-4)
  - IsoDep (ISO 14443-4)
  - NFC Forum Tag Types 1, 2, 3, 4, 5 (defined by the NFC Forum)
- [SR] STRONGLY RECOMMENDED to be capable of reading and writing NDEF messages as well as raw data via the following NFC standards. Note that while the NFC standards are stated as STRONGLY RECOMMENDED, the Compatibility Definition for a future version is planned to change these to MUST. These standards are optional in this version but will be required in future versions. Existing and new devices that run this version of Android are very strongly encouraged to meet these requirements now so they will be able to upgrade to the future platform releases.
- [C-1-13] MUST poll for all supported technologies while in NFC discovery mode.
- SHOULD be in NFC discovery mode while the device is awake with the screen active and the lock-screen unlocked.
- SHOULD be capable of reading the barcode and URL (if encoded) of Thinfilm NFC Barcode products.

Note that publicly available links are not available for the JIS, ISO, and NFC Forum specifications cited above.

Android includes support for NFC Host Card Emulation (HCE) mode.

If device implementations include an NFC controller chipset capable of HCE (for NfcA and/or NfcB) and support Application ID (AID) routing, they:

- [C-2-1] MUST report the android.hardware.nfc.hce feature constant.
- [C-2-2] MUST support NFC HCE APIs as defined in the Android SDK.

If device implementations include an NFC controller chipset capable of HCE for NfcF, and implement the feature for third-party applications, they:

- [C-3-1] MUST report the android.hardware.nfc.hcef feature constant.
- [C-3-2] MUST implement the NfcF Card Emulation APIs as defined in the Android SDK.

If device implementations include general NFC support as described in this section and support MIFARE technologies (MIFARE Classic, MIFARE Ultralight, NDEF on MIFARE Classic) in the reader/writer role, they:

- [C-4-1] MUST implement the corresponding Android APIs as documented by the Android SDK.
- [C-4-2] MUST report the feature com.nxp.mifare from the android.content.pm.PackageManager.hasSystemFeature () method. Note that this is not a standard Android feature and as such does not appear as a constant in the android.content.pm.PackageManager class.

7.4.5. Networking protocols and APIs

7.4.5.1. Minimum Network Capability

Device implementations:

- [C-0-1] MUST include support for one or more forms of data networking. Specifically, device implementations MUST include support for at least one data standard capable of 200 Kbit/sec or greater. Examples of technologies that satisfy this requirement include EDGE, HSPA, EV-DO, 802.11g, Ethernet and Bluetooth PAN.
- SHOULD also include support for at least one common wireless data standard, such as 802.11 (Wi-Fi), when a physical networking standard (such as Ethernet) is the primary data connection.
- MAY implement more than one form of data connectivity.

7.4.5.2. IPv6

Device implementations:

- [C-0-2] MUST include an IPv6 networking stack and support IPv6 communication using the managed APIs, such as java.net.Socket and java.net.URLConnection , as well as the native APIs, such as AF_INET6 sockets.
- [C-0-3] MUST enable IPv6 by default.
- MUST ensure that IPv6 communication is as reliable as IPv4, for example:
  - [C-0-4] MUST maintain IPv6 connectivity in doze mode.
  - [C-0-5] Rate-limiting MUST NOT cause the device to lose IPv6 connectivity on any IPv6-compliant network that uses RA lifetimes of at least 180 seconds.
- [C-0-6] MUST provide third-party applications with direct IPv6 connectivity to the network when connected to an IPv6 network, without any form of address or port translation happening locally on the device. Both managed APIs such as Socket#getLocalAddress or Socket#getLocalPort) and NDK APIs such as getsockname() or IPV6_PKTINFO MUST return the IP address and port that is actually used to send and receive packets on the network and is visible as the source ip and port to internet (web) servers.

The required level of IPv6 support depends on the network type, as shown in the following requirements.

If device implementations support Wi-Fi, they:

- [C-1-1] MUST support dual-stack and IPv6-only operation on Wi-Fi.

If device implementations support Ethernet, they:

- [C-2-1] MUST support dual-stack and IPv6-only operation on Ethernet.

If device implementations support Cellular data, they:

- [C-3-1] MUST support IPv6 operation (IPv6-only and possibly dual-stack) on cellular.

If device implementations support more than one network type (e.g., Wi-Fi and cellular data), they:

- [C-4-1] MUST simultaneously meet the above requirements on each network when the device is simultaneously connected to more than one network type.

7.4.5.3. Captive Portals

A captive portal refers to a network that requires sign-in in order to obtain internet access.

If device implementations provide a complete implementation of the android.webkit.Webview API , they:

- [C-1-1] MUST provide a captive portal application to handle the intent ACTION_CAPTIVE_PORTAL_SIGN_IN and display the captive portal login page, by sending that intent, on call to the System API ConnectivityManager#startCaptivePortalApp(Network, Bundle) .
- [C-1-2] MUST perform detection of captive portals and support login through the captive portal application when the device is connected to any network type, including cellular/mobile network, WiFi, Ethernet or Bluetooth.
- [C-1-3] MUST support logging in to captive portals using cleartext DNS when the device is configured to use private DNS strict mode.
- [C-1-4] MUST use encrypted DNS as per the SDK documentation for

android.net.LinkProperties.getPrivateDnsServerName and android.net.LinkProperties.isPrivateDnsActive for all network traffic that is not explicitly communicating with the captive portal.

- [C-1-5] MUST ensure that, while the user is logging in to a captive portal, the default network used by applications (as returned by ConnectivityManager.getActiveNetwork , ConnectivityManager.registerDefaultNetworkCallback , and used by default by Java networking APIs such as java.net.Socket, and native APIs such as connect()) is any other available network that provides internet access, if available.

7.4.6. Sync Settings

Device implementations:

- [C-0-1] MUST have the master auto-sync setting on by default so that the method getMasterSyncAutomatically() returns “true”.

7.4.7. Data Saver

If device implementations include a metered connection, they are:

- [SR] STRONGLY RECOMMENDED to provide the data saver mode.

If device implementations provide the data saver mode, they:

- [C-1-1] MUST support all the APIs in the ConnectivityManager class as described in the SDK documentation

If device implementations do not provide the data saver mode, they:

- [C-2-1] MUST return the value RESTRICT_BACKGROUND_STATUS_DISABLED for ConnectivityManager.getRestrictBackgroundStatus()
- [C-2-2] MUST NOT broadcast ConnectivityManager.ACTION_RESTRICT_BACKGROUND_CHANGED .

7.4.8. Secure Elements

If device implementations support Open Mobile API -capable secure elements and make them available to third-party apps, they:

- [C-1-1] MUST enumerate the available secure elements readers via android.se.omapi.SEService.getReaders() API.
- [C-1-2] MUST declare the correct feature flags via android.hardware.se.omapi.uicc for the device with UICC-based secure elements, android.hardware.se.omapi.ese for the device with eSE-based secure elements and android.hardware.se.omapi.sd for the device with SD-based secure elements.

7.5. Cameras

If device implementations include at least one camera, they:

- [C-1-1] MUST declare the android.hardware.camera.any feature flag.
- [C-1-2] MUST be possible for an application to simultaneously allocate 3 RGBA_8888 bitmaps equal to the size of the images produced by the largest-resolution camera sensor on the device, while camera is open for the purpose of basic preview and still capture.
- [C-1-3] MUST ensure that the preinstalled default camera application handling intents MediaStore.ACTION_IMAGE_CAPTURE , MediaStore.ACTION_IMAGE_CAPTURE_SECURE , or MediaStore.ACTION_VIDEO_CAPTURE , is responsible for removing the user location in the image metadata before sending it to the receiving application when the receiving application does not have ACCESS_FINE_LOCATION .

7.5.1. Rear-Facing Camera

A rear-facing camera is a camera located on the side of the device opposite the display; that is, it images scenes on the far side of the device, like a traditional camera.

Device implementations:

- SHOULD include a rear-facing camera.

If device implementations include at least one rear-facing camera, they:

- [C-1-1] MUST report the feature flag android.hardware.camera and android.hardware.camera.any .
- [C-1-2] MUST have a resolution of at least 2 megapixels.
- SHOULD have either hardware auto-focus or software auto-focus implemented in the camera driver (transparent to application software).
- MAY have fixed-focus or EDOF (extended depth of field) hardware.
- MAY include a flash.

If the camera includes a flash:

- [C-2-1] the flash lamp MUST NOT be lit while an android.hardware.Camera.PreviewCallback instance has been registered on a Camera preview surface, unless the application has explicitly enabled the flash by enabling the FLASH_MODE_AUTO or FLASH_MODE_ON attributes of a Camera.Parameters object. Note that this constraint does not apply to the device's built-in system camera application, but only to third-party applications using Camera.PreviewCallback .

7.5.2. Front-Facing Camera

A front-facing camera is a camera located on the same side of the device as the display; that is, a camera typically used to image the user, such as for video conferencing and similar applications.

Device implementations:

- MAY include a front-facing camera.

If device implementations include at least one front-facing camera, they:

- [C-1-1] MUST report the feature flag android.hardware.camera.any and android.hardware.camera.front .
- [C-1-2] MUST have a resolution of at least VGA (640x480 pixels).
- [C-1-3] MUST NOT use a front-facing camera as the default for the Camera API and MUST NOT configure the API to treat a front-facing camera as the default rear-facing camera, even if it is the only camera on the device.
- [C-1-4] The camera preview MUST be mirrored horizontally relative to the orientation specified by the application when the current application has explicitly requested that the Camera display be rotated via a call to the android.hardware.Camera.setDisplayOrientation() method. Conversely, the preview MUST be mirrored along the device's default horizontal axis when the current application does not explicitly request that the Camera display be rotated via a call to the android.hardware.Camera.setDisplayOrientation() method.
- [C-1-5] MUST NOT mirror the final captured still image or video streams returned to application callbacks or committed to media storage.
- [C-1-6] MUST mirror the image displayed by the postview in the same manner as the camera preview image stream.
- MAY include features (such as auto-focus, flash, etc.) available to rear-facing cameras as described in section 7.5.1 .

If device implementations are capable of being rotated by user (such as automatically via an accelerometer or manually via user input):

- [C-2-1] The camera preview MUST be mirrored horizontally relative to the device's current orientation.

7.5.3. External Camera

Device implementations:

- MAY include support for an external camera that is not necessarily always connected.

If device implementations include support for an external camera, they:

- [C-1-1] MUST declare the platform feature flag android.hardware.camera.external and

android.hardware camera.any .

- [C-1-2] MUST support USB Video Class (UVC 1.0 or higher) if the external camera connects through the USB host port.
- [C-1-3] MUST pass camera CTS tests with a physical external camera device connected. Details of camera CTS testing are available at source.android.com .
- SHOULD support video compressions such as MJPEG to enable transfer of high-quality unencoded streams (i.e. raw or independently compressed picture streams).
- MAY support multiple cameras.
- MAY support camera-based video encoding.

If camera-based video encoding is supported:

- [C-2-1] A simultaneous unencoded / MJPEG stream (QVGA or greater resolution) MUST be accessible to the device implementation.

7.5.4. Camera API Behavior

Android includes two API packages to access the camera, the newer android.hardware.camera2 API expose lower-level camera control to the app, including efficient zero-copy burst/streaming flows and per-frame controls of exposure, gain, white balance gains, color conversion, denoising, sharpening, and more.

The older API package, android.hardware.Camera , is marked as deprecated in Android 5.0 but as it should still be available for apps to use. Android device implementations MUST ensure the continued support of the API as described in this section and in the Android SDK.

All features that are common between the deprecated android.hardware.Camera class and the newer android.hardware.camera2 package MUST have equivalent performance and quality in both APIs. For example, with equivalent settings, autofocus speed and accuracy must be identical, and the quality of captured images must be the same. Features that depend on the different semantics of the two APIs are not required to have matching speed or quality, but SHOULD match as closely as possible.

Device implementations MUST implement the following behaviors for the camera-related APIs, for all available cameras. Device implementations:

- [C-0-1] MUST use android.hardware.PixelFormat.YCbCr_420_SP for preview data provided to application callbacks when an application has never called android.hardware.Camera.Parameters.setPreviewFormat(int) .
- [C-0-2] MUST further be in the NV21 encoding format when an application registers an android.hardware.Camera.PreviewCallback instance and the system calls the onPreviewFrame() method and the preview format is YCbCr_420_SP, the data in the byte[] passed into onPreviewFrame() . That is, NV21 MUST be the default.
- [C-0-3] MUST support the YV12 format (as denoted by the android.graphics.ImageFormat.YV12 constant) for camera previews for both front- and rear-facing cameras for android.hardware.Camera . (The hardware video encoder and camera may use any native pixel format, but the device implementation MUST support conversion to YV12.)
- [C-0-4] MUST support the android.hardware.ImageFormat.YUV_420_888 and android.hardware.ImageFormat.JPEG formats as outputs through the android.media.ImageReader API for android.hardware.camera2 devices that advertise REQUEST_AVAILABLE_CAPABILITIES_BACKWARD_COMPATIBLE capability in android.request.availableCapabilities .
- [C-0-5] MUST still implement the full Camera API included in the Android SDK documentation, regardless of whether the device includes hardware autofocus or other capabilities. For instance, cameras that lack autofocus MUST still call any registered android.hardware.Camera.AutoFocusCallback instances (even though this has no relevance to a non-autofocus camera.) Note that this does apply to front-facing cameras; for instance, even though most front-facing cameras do not support autofocus, the API callbacks must still be “faked” as described.
- [C-0-6] MUST recognize and honor each parameter name defined as a constant in the android.hardware.Camera.Parameters class and the android.hardware.camera2.CaptureRequest class. Conversely, device implementations MUST NOT honor or recognize string constants passed to the android.hardware.Camera.setParameters() method other than those documented as constants on the android.hardware.Camera.Parameters . That is, device implementations MUST support all standard Camera parameters if the hardware allows, and MUST NOT support custom Camera parameter types. For instance, device implementations that support image capture using high dynamic range (HDR) imaging techniques MUST support camera parameter Camera.SCENE_MODE_HDR .

- [C-0-7] MUST report the proper level of support with the android.info.supportedHardwareLevel property as described in the Android SDK and report the appropriate framework feature flags .
- [C-0-8] MUST also declare its individual camera capabilities of android.hardware.camera2 via the android.request.availableCapabilities property and declare the appropriate feature flags ; MUST define the feature flag if any of its attached camera devices supports the feature.
- [C-0-9] MUST broadcast the Camera.ACTION_NEW_PICTURE intent whenever a new picture is taken by the camera and the entry of the picture has been added to the media store.
- [C-0-10] MUST broadcast the Camera.ACTION_NEW_VIDEO intent whenever a new video is recorded by the camera and the entry of the picture has been added to the media store.
- [C-0-11] MUST have all cameras accessible via the deprecated android.hardware.Camera API also accessible via the android.hardware.camera2 API.
- [C-0-12] MUST ensure that the facial appearance is NOT altered, including but not limited to altering facial geometry, facial skin tone, or facial skin smoothening for any android.hardware.camera2 or android.hardware.Camera API.
- [C-SR] For devices with multiple RGB cameras facing in the same direction, are STRONGLY RECOMMENDED to support a logical camera device that lists capability CameraMetadata.REQUEST_AVAILABLE_CAPABILITIES_LOGICAL_MULTI_CAMERA , consisting of all of the RGB cameras facing that direction as physical sub-devices.

If device implementations provide a proprietary camera API to 3rd-party apps, they:

- [C-1-1] MUST implement such a camera API using android.hardware.camera2 API.
- MAY provide vendor tags and/or extensions to android.hardware.camera2 API.

7.5.5. Camera Orientation

If device implementations have a front- or a rear-facing camera, such camera(s):

- [C-1-1] MUST be oriented so that the long dimension of the camera aligns with the screen's long dimension. That is, when the device is held in the landscape orientation, cameras MUST capture images in the landscape orientation. This applies regardless of the device's natural orientation; that is, it applies to landscape-primary devices as well as portrait-primary devices.

7.6. Memory and Storage

7.6.1. Minimum Memory and Storage

Device implementations:

- [C-0-1] MUST include a Download Manager that applications MAY use to download data files and they MUST be capable of downloading individual files of at least 100MB in size to the default "cache" location.

7.6.2. Application Shared Storage

Device implementations:

- [C-0-1] MUST offer storage to be shared by applications, also often referred as "shared external storage", "application shared storage" or by the Linux path "/sdcard" it is mounted on.
- [C-0-2] MUST be configured with shared storage mounted by default, in other words "out of the box", regardless of whether the storage is implemented on an internal storage component or a removable storage medium (e.g. Secure Digital card slot).
- [C-0-3] MUST mount the application shared storage directly on the Linux path sdcard or include a Linux symbolic link from sdcard to the actual mount point.
- [C-0-4] MUST enable scoped storage by default for all apps targeting API level 29 or above, except in the following situation:
  - When the app has requested android:requestLegacyExternalStorage="true" in their manifest.
- [C-0-5] MUST redact location metadata, such as GPS Exif tags, stored in media files when those files are accessed through MediaStore , except when the calling app holds the ACCESS_MEDIA_LOCATION permission.

Device implementations MAY meet the above requirements using either of the following:

- User-accessible removable storage, such as a Secure Digital (SD) card slot.
- A portion of the internal (non-removable) storage as implemented in the Android Open Source Project (AOSP).

If device implementations use removable storage to satisfy the above requirements, they:

- [C-1-1] MUST implement a toast or pop-up user interface warning the user when there is no storage medium inserted in the slot.
- [C-1-2] MUST include a FAT-formatted storage medium (e.g. SD card) or show on the box and other material available at time of purchase that the storage medium has to be purchased separately.

If device implementations use a portion of the non-removable storage to satisfy the above requirements, they:

- SHOULD use the AOSP implementation of the internal application shared storage.
- MAY share the storage space with the application private data.

If device implementations have a USB port with USB peripheral mode support, they:

- [C-3-1] MUST provide a mechanism to access the data on the application shared storage from a host computer.
- SHOULD expose content from both storage paths transparently through Android's media scanner service and android.provider.MediaStore .
- MAY use USB mass storage, but SHOULD use Media Transfer Protocol to satisfy this requirement.

If device implementations have a USB port with USB peripheral mode and support Media Transfer Protocol, they:

- SHOULD be compatible with the reference Android MTP host, Android File Transfer .
- SHOULD report a USB device class of 0x00.
- SHOULD report a USB interface name of 'MTP'.

7.6.3. Adoptable Storage

If the device is expected to be mobile in nature unlike Television, device implementations are:

- [SR] STRONGLY RECOMMENDED to implement the adoptable storage in a long-term stable location, since accidentally disconnecting them can cause data loss/corruption.

If the removable storage device port is in a long-term stable location, such as within the battery compartment or other protective cover, device implementations are:

- [SR] STRONGLY RECOMMENDED to implement adoptable storage .

7.7. USB

If device implementations have a USB port, they:

- SHOULD support USB peripheral mode and SHOULD support USB host mode.

7.7.1. USB peripheral mode

If device implementations include a USB port supporting peripheral mode:

- [C-1-1] The port MUST be connectable to a USB host that has a standard type-A or type-C USB port.
- [C-1-2] MUST report the correct value of iSerialNumber in USB standard device descriptor through android.os.Build.SERIAL .
- [C-1-3] MUST detect 1.5A and 3.0A chargers per the Type-C resistor standard and MUST detect changes in the advertisement if they support Type-C USB.
- [SR] The port SHOULD use micro-B, micro-AB or Type-C USB form factor. Existing and

new Android devices are **STRONGLY RECOMMENDED to meet these requirements** so they will be able to upgrade to the future platform releases.

- [SR] The port SHOULD be located on the bottom of the device (according to natural orientation) or enable software screen rotation for all apps (including home screen), so that the display draws correctly when the device is oriented with the port at bottom. Existing and new Android devices are **STRONGLY RECOMMENDED to meet these requirements** so they will be able to upgrade to future platform releases.
- [SR] SHOULD implement support to draw 1.5 A current during HS chirp and traffic as specified in the USB Battery Charging specification, revision 1.2 . Existing and new Android devices are **STRONGLY RECOMMENDED to meet these requirements** so they will be able to upgrade to the future platform releases.
- [SR] STRONGLY RECOMMENDED to not support proprietary charging methods that modify Vbus voltage beyond default levels, or alter sink/source roles as such may result in interoperability issues with the chargers or devices that support the standard USB Power Delivery methods. While this is called out as "STRONGLY RECOMMENDED", in future Android versions we might REQUIRE all type-C devices to support full interoperability with standard type-C chargers.
- [SR] STRONGLY RECOMMENDED to support Power Delivery for data and power role swapping when they support Type-C USB and USB host mode.
- SHOULD support Power Delivery for high-voltage charging and support for Alternate Modes such as display out.
- SHOULD implement the Android Open Accessory (AOA) API and specification as documented in the Android SDK documentation.

If device implementations include a USB port and implement the AOA specification, they:

- [C-2-1] MUST declare support for the hardware feature android.hardware.usb.accessory .
- [C-2-2] The USB mass storage class MUST include the string "android" at the end of the interface description iInterface string of the USB mass storage
- SHOULD NOT implement AOAv2 audio documented in the Android Open Accessory Protocol 2.0 documentation. AOAv2 audio is deprecated as of Android version 8.0 (API level 26).

7.7.2. USB host mode

If device implementations include a USB port supporting host mode, they:

- [C-1-1] MUST implement the Android USB host API as documented in the Android SDK and MUST declare support for the hardware feature android.hardware.usb.host .
- [C-1-2] MUST implement support to connect standard USB peripherals, in other words, they MUST either:
  - Have an on-device type C port or ship with cable(s) adapting an on-device proprietary port to a standard USB type-C port (USB Type-C device).
  - Have an on-device type A or ship with cable(s) adapting an on-device proprietary port to a standard USB type-A port.
  - Have an on-device micro-AB port, which SHOULD ship with a cable adapting to a standard type-A port.
- [C-1-3] MUST NOT ship with an adapter converting from USB type A or micro-AB ports to a type-C port (receptacle).
- [C-SR] Are STRONGLY RECOMMENDED to implement the USB audio class as documented in the Android SDK documentation.
- SHOULD support charging the connected USB peripheral device while in host mode; advertising a source current of at least 1.5A as specified in the Termination Parameters section of the USB Type-C Cable and Connector Specification Revision 1.2 for USB Type-C connectors or using Charging Downstream Port(CDP) output current range as specified in the USB Battery Charging specifications, revision 1.2 for Micro-AB connectors.
- SHOULD implement and support USB Type-C standards.

If device implementations include a USB port supporting host mode and the USB audio class, they:

- [C-2-1] MUST support the USB HID class .
- [C-2-2] MUST support the detection and mapping of the following HID data fields specified in the USB HID Usage Tables and the Voice Command Usage Request to the KeyEvent constants as below:
  - Usage Page (0xC) Usage ID (0x0CD): KEYCODE_MEDIA_PLAY_PAUSE

- Usage Page (0xC) Usage ID (0x0E9): KEYCODE_VOLUME_UP
- Usage Page (0xC) Usage ID (0x0EA): KEYCODE_VOLUME_DOWN
- Usage Page (0xC) Usage ID (0x0CF): KEYCODE_VOICE_ASSIST

If device implementations include a USB port supporting host mode and the Storage Access Framework (SAF), they:

- [C-3-1] MUST recognize any remotely connected MTP (Media Transfer Protocol) devices and make their contents accessible through the ACTION_GET_CONTENT , ACTION_OPEN_DOCUMENT , and ACTION_CREATE_DOCUMENT intents. .

If device implementations include a USB port supporting host mode and USB Type-C, they:

- [C-4-1] MUST implement Dual Role Port functionality as defined by the USB Type-C specification (section 4.5.1.3.3).
- [SR] STRONGLY RECOMMENDED to support DisplayPort, SHOULD support USB SuperSpeed Data Rates, and are STRONGLY RECOMMENDED to support Power Delivery for data and power role swapping.
- [SR] STRONGLY RECOMMENDED to NOT support Audio Adapter Accessory Mode as described in the Appendix A of the USB Type-C Cable and Connector Specification Revision 1.2 .
- SHOULD implement the Try.* model that is most appropriate for the device form factor. For example a handheld device SHOULD implement the Try.SNK model.

7.8. Audio

7.8.1. Microphone

If device implementations include a microphone, they:

- [C-1-1] MUST report the android.hardware.microphone feature constant.
- [C-1-2] MUST meet the audio recording requirements in section 5.4 .
- [C-1-3] MUST meet the audio latency requirements in section 5.6 .
- [SR] Are STRONGLY RECOMMENDED to support near-ultrasound recording as described in section 7.8.3 .

If device implementations omit a microphone, they:

- [C-2-1] MUST NOT report the android.hardware.microphone feature constant.
- [C-2-2] MUST implement the audio recording API at least as no-ops, per section 7 .

7.8.2. Audio Output

If device implementations include a speaker or an audio/multimedia output port for an audio output peripheral such as a 4 conductor 3.5mm audio jack or USB host mode port using USB audio class , they:

- [C-1-1] MUST report the android.hardware.audio.output feature constant.
- [C-1-2] MUST meet the audio playback requirements in section 5.5 .
- [C-1-3] MUST meet the audio latency requirements in section 5.6 .
- [SR] STRONGLY RECOMMENDED to support near-ultrasound playback as described in section 7.8.3 .

If device implementations do not include a speaker or audio output port, they:

- [C-2-1] MUST NOT report the android.hardware.audio.output feature.
- [C-2-2] MUST implement the Audio Output related APIs as no-ops at least.

For the purposes of this section, an "output port" is a physical interface such as a 3.5mm audio jack, HDMI, or USB host mode port with USB audio class. Support for audio output over radio-based protocols such as Bluetooth, WiFi, or cellular network does not qualify as including an "output port".

7.8.2.1. Analog Audio Ports

In order to be compatible with the headsets and other audio accessories using the 3.5mm audio plug across the Android ecosystem, if device implementations include one or more analog audio ports, they:

- [C-SR] Are STRONGLY RECOMMENDED to include at least one of the audio port(s) to be a 4 conductor 3.5mm audio jack.

If device implementations have a 4 conductor 3.5mm audio jack, they:

- [C-1-1] MUST support audio playback to stereo headphones and stereo headsets with a microphone.
- [C-1-2] MUST support TRRS audio plugs with the CTIA pin-out order.
- [C-1-3] MUST support the detection and mapping to the keycodes for the following 3 ranges of equivalent impedance between the microphone and ground conductors on the audio plug:
  - **70 ohm or less** : KEYCODE_HEADSETHOOK
  - **210-290 ohm** : KEYCODE_VOLUME_UP
  - **360-680 ohm** : KEYCODE_VOLUME_DOWN
- [C-1-4] MUST trigger ACTION_HEADSET_PLUG upon a plug insert, but only after all contacts on plug are touching their relevant segments on the jack.
- [C-1-5] MUST be capable of driving at least 150mV ± 10% of output voltage on a 32 ohm speaker impedance.
- [C-1-6] MUST have a microphone bias voltage between 1.8V ~ 2.9V.
- [C-1-7] MUST detect and map to the keycode for the following range of equivalent impedance between the microphone and ground conductors on the audio plug:
  - **110-180 ohm:** KEYCODE_VOICE_ASSIST
- [C-SR] Are STRONGLY RECOMMENDED to support audio plugs with the OMTP pin-out order.
- [C-SR] Are STRONGLY RECOMMEND to support audio recording from stereo headsets with a microphone.

If device implementations have a 4 conductor 3.5mm audio jack and support a microphone, and broadcast the android.intent.action.HEADSET_PLUG with the extra value microphone set as 1, they:

- [C-2-1] MUST support the detection of microphone on the plugged in audio accessory.

7.8.2.2. Digital Audio Ports

In order to be compatible with the headsets and other audio accessories using USB-C connectors and implementing (USB audio class) across the Android ecosystem as defined in Android USB headset specification .

See Section 2.2.1 for device-specific requirements.

7.8.3. Near-Ultrasound

Near-Ultrasound audio is the 18.5 kHz to 20 kHz band.

Device implementations:

- MUST correctly report the support of near-ultrasound audio capability via the AudioManager.getProperty API as follows:

If PROPERTY_SUPPORT_MIC_NEAR_ULTRASOUND is "true", the following requirements MUST be met by the VOICE_RECOGNITION and UNPROCESSED audio sources:

- [C-1-1] The microphone's mean power response in the 18.5 kHz to 20 kHz band MUST be no more than 15 dB below the response at 2 kHz.
- [C-1-2] The microphone's unweighted signal to noise ratio over 18.5 kHz to 20 kHz for a 19 kHz tone at -26 dBFS MUST be no lower than 50 dB.

If PROPERTY_SUPPORT_SPEAKER_NEAR_ULTRASOUND is "true":

- [C-2-1] The speaker's mean response in 18.5 kHz - 20 kHz MUST be no lower than 40 dB below the response at 2 kHz.

7.8.4. Signal Integrity

Device implementations: * SHOULD provide a glitch-free audio signal path for both input and output streams on handheld devices, as defined by zero glitches measured during a test of one minute per path. Test using [OboeTester] (https://github.com/google/oboe/tree/master/apps/OboeTester) “Automated Glitch Test”.

The test requires an [audio loopback dongle] (https://source.android.com/devices/audio/latency/loopback), used directly in a 3.5mm jack, and/or in combination with a USB-C to 3.5mm adapter. All audio output ports SHOULD be tested.

OboeTester currently supports AAudio paths, so the following combinations SHOULD be tested for glitches using AAudio:

Perf Mode	Sharing	Out Sample Rate	In Chans	Out Chans
LOW_LATENCY	EXCLUSIVE	UNSPECIFIED	1	2
LOW_LATENCY	EXCLUSIVE	UNSPECIFIED	2	1
LOW_LATENCY	SHARED	UNSPECIFIED	1	2
LOW_LATENCY	SHARED	UNSPECIFIED	2	1
NONE	SHARED	48000	1	2
NONE	SHARED	48000	2	1
NONE	SHARED	44100	1	2
NONE	SHARED	44100	2	1
NONE	SHARED	16000	1	2
NONE	SHARED	16000	2	1

A reliable stream SHOULD meet the following criteria for Signal to Noise Ratio (SNR) and Total Harmonic Distortion (THD) for 2000 Hz sine.

Transducer	THD	SNR
primary built-in speaker, measured using an external reference microphone	< 3.0%	>= 50 dB
primary built-in microphone, measured using an external reference speaker	< 3.0%	>= 50 dB
built-in analog 3.5 mm jacks, tested using loopback adapter	< 1%	>= 60 dB
USB adapters supplied with the phone, tested using loopback adapter	< 1.0%	>= 60 dB

7.9. Virtual Reality

Android includes APIs and facilities to build "Virtual Reality" (VR) applications including high quality mobile VR experiences. Device implementations MUST properly implement these APIs and behaviors, as detailed in this section.

7.9.1. Virtual Reality Mode

Android includes support for VR Mode , a feature which handles stereoscopic rendering of notifications and disables monocular system UI components while a VR application has user focus.

7.9.2. Virtual Reality Mode - High Performance

If device implementations support VR mode, they:

- [C-1-1] MUST have at least 2 physical cores.
- [C-1-2] MUST declare the android.hardware.vr.high_performance feature.
- [C-1-3] MUST support sustained performance mode.
- [C-1-4] MUST support OpenGL ES 3.2.
- [C-1-5] MUST support android.hardware.vulkan.level 0.
- SHOULD support android.hardware.vulkan.level 1 or higher.
- [C-1-6] MUST implement EGL_KHR_mutable_render_buffer , EGL_ANDROID_front_buffer_auto_refresh , EGL_ANDROID_get_native_client_buffer , EGL_KHR_fence_sync , EGL_KHR_wait_sync , EGL_IMG_context_priority , EGL_EXT_protected_content , EGL_EXT_image_gl_colorspace , and expose the extensions in the list of available EGL extensions.

- [C-1-8] MUST implement GL_EXT_multisampled_render_to_texture2 , GL_OVR_multiview , GL_OVR_multiview2 , GL_OVR_multiview_multisampled_render_to_texture , GL_EXT_protected_textures , and expose the extensions in the list of available GL extensions.
- [C-SR] Are STRONGLY RECOMMENDED to implement GL_EXT_external_buffer , GL_EXT_EGL_image_array , and expose the extensions in the list of available GL extensions.
- [C-SR] Are STRONGLY RECOMMENDED to support Vulkan 1.1.
- [C-SR] Are STRONGLY RECOMMENDED to implement VK_ANDROID_external_memory_android_hardware_buffer , VK_GOOGLE_display_timing , VK_KHR_shared_presentable_image , and expose it in the list of available Vulkan extensions.
- [C-SR] Are STRONGLY RECOMMENDED to expose at least one Vulkan queue family where flags contain both VK_QUEUE_GRAPHICS_BIT and VK_QUEUE_COMPUTE_BIT , and queueCount is at least 2.
- [C-1-7] The GPU and display MUST be able to synchronize access to the shared front buffer such that alternating-eye rendering of VR content at 60fps with two render contexts will be displayed with no visible tearing artifacts.
- [C-1-9] MUST implement support for AHardwareBuffer flags AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER , AHARDWAREBUFFER_USAGE_SENSOR_DIRECT_DATA and AHARDWAREBUFFER_USAGE_PROTECTED_CONTENT as described in the NDK.
- [C-1-10] MUST implement support for AHardwareBuffer s with any combination of the usage flags AHARDWAREBUFFER_USAGE_GPU_COLOR_OUTPUT , AHARDWAREBUFFER_USAGE_GPU_SAMPLED_IMAGE , AHARDWAREBUFFER_USAGE_PROTECTED_CONTENT for at least the following formats: AHARDWAREBUFFER_FORMAT_R5G6B5_UNORM , AHARDWAREBUFFER_FORMAT_R8G8B8A8_UNORM , AHARDWAREBUFFER_FORMAT_R10G10B10A2_UNORM , AHARDWAREBUFFER_FORMAT_R16G16B16A16_FLOAT .
- [C-SR] Are STRONGLY RECOMMENDED to support the allocation of AHardwareBuffer s with more than one layer and flags and formats specified in C-1-10.
- [C-1-11] MUST support H.264 decoding at least 3840 x 2160 at 30fps, compressed to an average of 40Mbps (equivalent to 4 instances of 1920 x1080 at 30 fps-10 Mbps or 2 instances of 1920 x 1080 at 60 fps-20 Mbps).
- [C-1-12] MUST support HEVC and VP9, MUST be capable of decoding at least 1920 x 1080 at 30 fps compressed to an average of 10 Mbps and SHOULD be capable of decoding 3840 x 2160 at 30 fps-20 Mbps (equivalent to 4 instances of 1920 x 1080 at 30 fps-5 Mbps).
- [C-1-13] MUST support HardwarePropertiesManager.getDeviceTemperatures API and return accurate values for skin temperature.
- [C-1-14] MUST have an embedded screen, and its resolution MUST be at least 1920 x 1080.
- [C-SR] Are STRONGLY RECOMMENDED to have a display resolution of at least 2560 x 1440.
- [C-1-15] The display MUST update at least 60 Hz while in VR Mode.
- [C-1-17] The display MUST support a low-persistence mode with ≤ 5 milliseconds persistence, persistence being defined as the amount of time for which a pixel is emitting light.
- [C-1-18] MUST support Bluetooth 4.2 and Bluetooth LE Data Length Extension section 7.4.3 .
- [C-1-19] MUST support and properly report Direct Channel Type for all of the following default sensor types:
  - TYPE_ACCELEROMETER
  - TYPE_ACCELEROMETER_UNCALIBRATED
  - TYPE_GYROSCOPE
  - TYPE_GYROSCOPE_UNCALIBRATED
  - TYPE_MAGNETIC_FIELD
  - TYPE_MAGNETIC_FIELD_UNCALIBRATED
- [C-SR] Are STRONGLY RECOMMENDED to support the TYPE_HARDWARE_BUFFER direct channel type for all Direct Channel Types listed above.
- [C-1-21] MUST meet the gyroscope, accelerometer, and magnetometer related requirements for android.hardware.hifi_sensors , as specified in section 7.3.9 .
- [C-SR] Are STRONGLY RECOMMENDED to support the android.hardware.sensor.hifi_sensors

feature.

- [C-1-22] MUST have end-to-end motion to photon latency not higher than 28 milliseconds.
- [C-SR] Are STRONGLY RECOMMENDED to have end-to-end motion to photon latency not higher than 20 milliseconds.
- [C-1-23] MUST have first-frame ratio, which is the ratio between the brightness of pixels on the first frame after a transition from black to white and the brightness of white pixels in steady state, of at least 85%.
- [C-SR] Are STRONGLY RECOMMENDED to have first-frame ratio of at least 90%.
- MAY provide an exclusive core to the foreground application and MAY support the `Process.getExclusiveCores` API to return the numbers of the cpu cores that are exclusive to the top foreground application.

If exclusive core is supported, then the core:

- [C-2-1] MUST not allow any other userspace processes to run on it (except device drivers used by the application), but MAY allow some kernel processes to run as necessary.

7.10. Haptics

See Section 2.2.1 for device-specific requirements.

8. Performance and Power

Some minimum performance and power criteria are critical to the user experience and impact the baseline assumptions developers would have when developing an app.

8.1. User Experience Consistency

A smooth user interface can be provided to the end user if there are certain minimum requirements to ensure a consistent frame rate and response times for applications and games. Device implementations, depending on the device type, MAY have measurable requirements for the user interface latency and task switching as described in section 2 .

8.2. File I/O Access Performance

Providing a common baseline for a consistent file access performance on the application private data storage (`/data` partition) allows app developers to set a proper expectation that would help their software design. Device implementations, depending on the device type, MAY have certain requirements described in section 2 for the following read and write operations:

- **Sequential write performance** . Measured by writing a 256MB file using 10MB write buffer.
- **Random write performance** . Measured by writing a 256MB file using 4KB write buffer.
- **Sequential read performance** . Measured by reading a 256MB file using 10MB write buffer.
- **Random read performance** . Measured by reading a 256MB file using 4KB write buffer.

8.3. Power-Saving Modes

If device implementations include features to improve device power management that are included in AOSP (e.g. App Standby Bucket, Doze) or extend the features that do not apply harder restrictions than the Rare App Standby Bucket , they:

- [C-1-1] MUST NOT deviate from the AOSP implementation for the triggering, maintenance, wakeup algorithms and the use of global system settings of App Standby and Doze power-saving modes.
- [C-1-2] MUST NOT deviate from the AOSP implementation for the use of global settings to manage the throttling of jobs, alarm and network for apps in each bucket for App standby.
- [C-1-3] MUST NOT deviate from the AOSP implementation for the number of the App Standby Buckets used for App Standby.
- [C-1-4] MUST implement App Standby Buckets and Doze as described in Power Management .
- [C-1-5] MUST return `true` for `PowerManager.isPowerSaveMode()` when the device is on power save mode.

- [C-SR] Are STRONGLY RECOMMENDED to provide user affordance to enable and disable the battery saver feature.
- [C-SR] Are STRONGLY RECOMMENDED to provide user affordance to display all Apps that are exempted from App Standby and Doze power-saving modes.

If device implementations extend power management features that are included in AOSP and that extension applies more stringent restrictions than the Rare App Standby Bucket , refer to section 3.5.1 .

In addition to the power-saving modes, Android device implementations MAY implement any or all of the 4 sleeping power states as defined by the Advanced Configuration and Power Interface (ACPI).

If device implementations implement S4 power states as defined by the ACPI, they:

- [C-1-1] MUST enter this state only after the user has taken an explicit action to put the device in an inactive state (e.g. by closing a lid that is physically part of the device or turning off a vehicle or television) and before the user re-activates the device (e.g. by opening the lid or turning the vehicle or television back on).

If device implementations implement S3 power states as defined by the ACPI, they:

- [C-2-1] MUST meet C-1-1 above, or, MUST enter S3 state only when third-party applications do not need the system resources (e.g. the screen, CPU).

  Conversely, MUST exit from S3 state when third-party applications need the system resources, as described on this SDK.

  For example, while the third-party applications request to keep the screen on through FLAG_KEEP_SCREEN_ON or keep CPU running through PARTIAL_WAKE_LOCK , the device MUST NOT enter S3 state unless, as described in C-1-1, the user has taken explicit action to put the device in an inactive state. Conversely, at a time when a task that third-party apps implement through JobScheduler is triggered or Firebase Cloud Messaging is delivered to third-party apps, the device MUST exit the S3 state unless the user has put the device in an inactive state. These are not comprehensive examples and AOSP implements extensive wake-up signals that trigger a wakeup from this state.

8.4. Power Consumption Accounting

A more accurate accounting and reporting of the power consumption provides the app developer both the incentives and the tools to optimize the power usage pattern of the application.

Device implementations:

- [SR] STRONGLY RECOMMENDED to provide a per-component power profile that defines the current consumption value for each hardware component and the approximate battery drain caused by the components over time as documented in the Android Open Source Project site.
- [SR] STRONGLY RECOMMENDED to report all power consumption values in milliampere hours (mAh).
- [SR] STRONGLY RECOMMENDED to report CPU power consumption per each process's UID. The Android Open Source Project meets the requirement through the uid_cputime kernel module implementation.
- [SR] STRONGLY RECOMMENDED to make this power usage available via the adb shell dumpsys batterystats shell command to the app developer.
- SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.

8.5. Consistent Performance

Performance can fluctuate dramatically for high-performance long-running apps, either because of the other apps running in the background or the CPU throttling due to temperature limits. Android includes programmatic interfaces so that when the device is capable, the top foreground application can request that the system optimize the allocation of the resources to address such fluctuations.

Device implementations:

- [C-0-1] MUST report the support of Sustained Performance Mode accurately through the PowerManager.isSustainedPerformanceModeSupported() API method.
- SHOULD support Sustained Performance Mode.

If device implementations report support of Sustained Performance Mode, they:

- [C-1-1] MUST provide the top foreground application a consistent level of performance for at least 30 minutes, when the app requests it.
- [C-1-2] MUST honor the Window.setSustainedPerformanceMode() API and other related APIs.

If device implementations include two or more CPU cores, they:

- SHOULD provide at least one exclusive core that can be reserved by the top foreground application.

If device implementations support reserving one exclusive core for the top foreground application, they:

- [C-2-1] MUST report through the Process.getExclusiveCores() API method the ID numbers of the exclusive cores that can be reserved by the top foreground application.
- [C-2-2] MUST not allow any user space processes except the device drivers used by the application to run on the exclusive cores, but MAY allow some kernel processes to run as necessary.

If device implementations do not support an exclusive core, they:

- [C-3-1] MUST return an empty list through the Process.getExclusiveCores() API method.

9. Security Model Compatibility

Device implementations:

- [C-0-1] MUST implement a security model consistent with the Android platform security model as defined in Security and Permissions reference document in the APIs in the Android developer documentation.
- [C-0-2] MUST support installation of self-signed applications without requiring any additional permissions/certificates from any third parties/authorities. Specifically, compatible devices MUST support the security mechanisms described in the follow subsections.

9.1. Permissions

Device implementations:

- [C-0-1] MUST support the Android permissions model as defined in the Android developer documentation. Specifically, they MUST enforce each permission defined as described in the SDK documentation; no permissions may be omitted, altered, or ignored.
- MAY add additional permissions, provided the new permission ID strings are not in the android.* namespace.
- [C-0-2] Permissions with a protectionLevel of PROTECTION_FLAG_PRIVILEGED MUST only be granted to apps preinstalled in the privileged path(s) of the system image and within the subset of the explicitly whitelisted permissions for each app. The AOSP implementation meets this requirement by reading and honoring the whitelisted permissions for each app from the files in the etc/permissions/ path and using the system/priv-app path as the privileged path.

Permissions with a protection level of dangerous are runtime permissions. Applications with targetSdkVersion > 22 request them at runtime.
Device implementations:

- [C-0-3] MUST show a dedicated interface for the user to decide whether to grant the requested runtime permissions and also provide an interface for the user to manage runtime permissions.
- [C-0-4] MUST have one and only one implementation of both user interfaces.
- [C-0-5] MUST NOT grant any runtime permissions to preinstalled apps unless:
  - The user's consent can be obtained before the application uses it.
  - The runtime permissions are associated with an intent pattern for which the preinstalled application is set as the default handler.

- [C-0-6] MUST grant the android.permission.RECOVER_KEYSTORE permission only to system apps that register a properly secured Recovery Agent. A properly secured Recovery Agent is defined as an on-device software agent that synchronizes with an off-device remote storage, that is equipped with secure hardware with protection equivalent or stronger than what is described in Google Cloud Key Vault Service to prevent brute-force attacks on the lockscreen knowledge factor.

Device implementations:

- [C-0-7] MUST adhere to Android location permission properties when an app requests the location or physical activity data through standard Android API or proprietary mechanism. Such data includes but not limited to:
  - Device's location (e.g. latitude and longitude).
  - Information that can be used to determine or estimate the device's location (e.g. SSID, BSSID, Cell ID, or location of the network that the device is connected to).
  - User's physical activity or classification of the physical activity.

More specifically, device implementations:

* [C-0-8] MUST obtain user consent to allow an app to access the location or physical activity data.
* [C-0-9] MUST grant a runtime permission ONLY to the app that holds sufficient permission as described on SDK.
For example,

TelephonyManager#getServiceState requires android.permission.ACCESS_FINE_LOCATION).

Permissions can be marked as restricted altering their behavior.

- [C-0-10] Permissions marked with the flag hardRestricted MUST NOT be granted to an app unless:
  - An app APK file is in the system partition.
  - The user assigns a role that is associated with the hardRestricted permissions to an app.
  - The installer grants the hardRestricted to an app.
  - An app is granted the hardRestricted on an earlier Android version.
- [C-0-11] Apps holding a softRestricted permission MUST get only limited access and MUST NOT gain full access until whitelisted as described in the SDK, where full and limited access is defined for each softRestricted permission (for example, READ_EXTERNAL_STORAGE).

If device implementations provide a user affordance to choose which apps can draw on top of other apps with an activity that handles the ACTION_MANAGE_OVERLAY_PERMISSION intent, they:

- [C-2-1] MUST ensure that all activities with intent filters for the ACTION_MANAGE_OVERLAY_PERMISSION intent have the same UI screen, regardless of the initiating app or any information it provides.

9.2. UID and Process Isolation

Device implementations:

- [C-0-1] MUST support the Android application sandbox model, in which each application runs as a unique Unixstyle UID and in a separate process.
- [C-0-2] MUST support running multiple applications as the same Linux user ID, provided that the applications are properly signed and constructed, as defined in the Security and Permissions reference .

9.3. Filesystem Permissions

Device implementations:

- [C-0-1] MUST support the Android file access permissions model as defined in the Security and Permissions reference .

9.4. Alternate Execution Environments

Device implementations MUST keep consistency of the Android security and permission model, even if they include runtime environments that execute applications using some other software or technology than the Dalvik Executable Format or native code. In other words:

- [C-0-1] Alternate runtimes MUST themselves be Android applications, and abide by the standard Android security model, as described elsewhere in section 9 .
- [C-0-2] Alternate runtimes MUST NOT be granted access to resources protected by permissions not requested in the runtime's AndroidManifest.xml file via the < uses-permission > mechanism.
- [C-0-3] Alternate runtimes MUST NOT permit applications to make use of features protected by Android permissions restricted to system applications.
- [C-0-4] Alternate runtimes MUST abide by the Android sandbox model and installed applications using an alternate runtime MUST NOT reuse the sandbox of any other app installed on the device, except through the standard Android mechanisms of shared user ID and signing certificate.
- [C-0-5] Alternate runtimes MUST NOT launch with, grant, or be granted access to the sandboxes corresponding to other Android applications.
- [C-0-6] Alternate runtimes MUST NOT be launched with, be granted, or grant to other applications any privileges of the superuser (root), or of any other user ID.
- [C-0-7] When the .apk files of alternate runtimes are included in the system image of device implementations, it MUST be signed with a key distinct from the key used to sign other applications included with the device implementations.
- [C-0-8] When installing applications, alternate runtimes MUST obtain user consent for the Android permissions used by the application.
- [C-0-9] When an application needs to make use of a device resource for which there is a corresponding Android permission (such as Camera, GPS, etc.), the alternate runtime MUST inform the user that the application will be able to access that resource.
- [C-0-10] When the runtime environment does not record application capabilities in this manner, the runtime environment MUST list all permissions held by the runtime itself when installing any application using that runtime.
- Alternate runtimes SHOULD install apps via the PackageManager into separate Android sandboxes (Linux user IDs, etc.).
- Alternate runtimes MAY provide a single Android sandbox shared by all applications using the alternate runtime.

9.5. Multi-User Support

Android includes support for multiple users and provides support for full user isolation.

- Device implementations MAY but SHOULD NOT enable multi-user if they use removable media for primary external storage.

If device implementations include multiple users, they:

- [C-1-1] MUST meet the following requirements related to multi-user support .
- [C-1-2] MUST, for each user, implement a security model consistent with the Android platform security model as defined in Security and Permissions reference document in the APIs.
- [C-1-3] MUST have separate and isolated shared application storage (a.k.a. /sdcard) directories for each user instance.
- [C-1-4] MUST ensure that applications owned by and running on behalf a given user cannot list, read, or write to the files owned by any other user, even if the data of both users are stored on the same volume or filesystem.
- [C-1-5] MUST encrypt the contents of the SD card when multiuser is enabled using a key stored only on non-removable media accessible only to the system if device implementations use removable media for the external storage APIs. As this will make the media unreadable by a host PC, device implementations will be required to switch to MTP or a similar system to provide host PCs with access to the current user's data.

If device implementations include multiple users and do not declare the android.hardware.telephony

feature flag, they:

- [C-2-1] MUST support restricted profiles, a feature that allows device owners to manage additional users and their capabilities on the device. With restricted profiles, device owners can quickly set up separate environments for additional users to work in, with the ability to manage finer-grained restrictions in the apps that are available in those environments.

If device implementations include multiple users and declare the android.hardware.telephony feature flag, they:

- [C-3-1] MUST NOT support restricted profiles but MUST align with the AOSP implementation of controls to enable /disable other users from accessing the voice calls and SMS.

9.6. Premium SMS Warning

Android includes support for warning users of any outgoing premium SMS message . Premium SMS messages are text messages sent to a service registered with a carrier that may incur a charge to the user.

If device implementations declare support for android.hardware.telephony , they:

- [C-1-1] MUST warn users before sending a SMS message to numbers identified by regular expressions defined in /data/misc/sms/codes.xml file in the device. The upstream Android Open Source Project provides an implementation that satisfies this requirement.

9.7. Security Features

Device implementations MUST ensure compliance with security features in both the kernel and platform as described below.

The Android Sandbox includes features that use the Security-Enhanced Linux (SELinux) mandatory access control (MAC) system, seccomp sandboxing, and other security features in the Linux kernel. Device implementations:

- [C-0-1] MUST maintain compatibility with existing applications, even when SELinux or any other security features are implemented below the Android framework.
- [C-0-2] MUST NOT have a visible user interface when a security violation is detected and successfully blocked by the security feature implemented below the Android framework, but MAY have a visible user interface when an unblocked security violation occurs resulting in a successful exploit.
- [C-0-3] MUST NOT make SELinux or any other security features implemented below the Android framework configurable to the user or app developer.
- [C-0-4] MUST NOT allow an application that can affect another application through an API (such as a Device Administration API) to configure a policy that breaks compatibility.
- [C-0-5] MUST split the media framework into multiple processes so that it is possible to more narrowly grant access for each process as described in the Android Open Source Project site.
- [C-0-6] MUST implement a kernel application sandboxing mechanism which allows filtering of system calls using a configurable policy from multithreaded programs. The upstream Android Open Source Project meets this requirement through enabling the seccomp-BPF with threadgroup synchronization (TSYNC) as described in the Kernel Configuration section of source.android.com .

Kernel integrity and self-protection features are integral to Android security. Device implementations:

- [C-0-7] MUST implement kernel stack buffer overflow protection mechanisms. Examples of such mechanisms are CC_STACKPROTECTOR_REGULAR and CONFIG_CC_STACKPROTECTOR_STRONG .
- [C-0-8] MUST implement strict kernel memory protections where executable code is read-only, read-only data is non-executable and non-writable, and writable data is non-executable (e.g. CONFIG_DEBUG_RODATA or CONFIG_STRICT_KERNEL_RWX).
- [C-0-9] MUST implement static and dynamic object size bounds checking of copies between user-space and kernel-space (e.g. CONFIG_HARDENED_USERCOPY) on devices originally shipping with API level 28 or higher.
- [C-0-10] MUST NOT execute user-space memory when executing in the kernel mode (e.g.

hardware PXN, or emulated via CONFIG_CPU_SW_DOMAIN_PAN or CONFIG_ARM64_SW_TTBR0_PAN) on devices originally shipping with API level 28 or higher.

- [C-0-11] MUST NOT read or write user-space memory in the kernel outside of normal usercopy access APIs (e.g. hardware PAN, or emulated via CONFIG_CPU_SW_DOMAIN_PAN or CONFIG_ARM64_SW_TTBR0_PAN) on devices originally shipping with API level 28 or higher.
- [C-0-12] MUST implement kernel page table isolation if the hardware is vulnerable to CVE-2017-5754 on all devices originally shipping with API level 28 or higher (e.g. CONFIG_PAGE_TABLE_ISOLATION or CONFIG_UNMAP_KERNEL_AT_EL0).
- [C-0-13] MUST implement branch prediction hardening if the hardware is vulnerable to CVE-2017-5715 on all devices originally shipping with API level 28 or higher (e.g. CONFIG_HARDEN_BRANCH_PREDICTOR).
- [SR] STRONGLY RECOMMENDED to keep kernel data which is written only during initialization marked read-only after initialization (e.g. __ro_after_init).
- [C-SR] Are STRONGLY RECOMMENDED to randomize the layout of the kernel code and memory, and to avoid exposures that would compromise the randomization (e.g. CONFIG_RANDOMIZE_BASE with bootloader entropy via the /chosen/kaslr-seed Device Tree node or EFI_RNG_PROTOCOL).
- [C-SR] Are STRONGLY RECOMMENDED to enable control flow integrity (CFI) in the kernel to provide additional protection against code-reuse attacks (e.g. CONFIG_CFI_CLANG and CONFIG_SHADOW_CALL_STACK).
- [C-SR] Are STRONGLY RECOMMENDED not to disable Control-Flow Integrity (CFI), Shadow Call Stack (SCS) or Integer Overflow Sanitization (IntSan) on components that have it enabled.
- [C-SR] Are STRONGLY RECOMMENDED to enable CFI, SCS, and IntSan for any additional security-sensitive userspace components as explained in CFI and IntSan .
- [C-SR] Are STRONGLY RECOMMENDED to enable stack initialization in the kernel to prevent uses of uninitialized local variables (CONFIG_INIT_STACK_ALL or CONFIG_INIT_STACK_ALL_ZERO). Also, device implementations SHOULD NOT assume the value used by the compiler to initialize the locals.
- [C-SR] Are STRONGLY RECOMMENDED to enable heap initialization in the kernel to prevent uses of uninitialized heap allocations (CONFIG_INIT_ON_ALLOC_DEFAULT_ON) and they SHOULD NOT assume the value used by the kernel to initialize those allocations.

If device implementations use a Linux kernel, they:

- [C-1-1] MUST implement SELinux.
- [C-1-2] MUST set SELinux to global enforcing mode.
- [C-1-3] MUST configure all domains in enforcing mode. No permissive mode domains are allowed, including domains specific to a device/vendor.
- [C-1-4] MUST NOT modify, omit, or replace the neverallow rules present within the system/sepolicy folder provided in the upstream Android Open Source Project (AOSP) and the policy MUST compile with all neverallow rules present, for both AOSP SELinux domains as well as device/vendor specific domains.
- [C-1-5] MUST run third-party applications targeting API level 28 or higher in per-application SELinux sandboxes with per-app SELinux restrictions on each application's private data directory.
- SHOULD retain the default SELinux policy provided in the system/sepolicy folder of the upstream Android Open Source Project and only further add to this policy for their own device-specific configuration.

If device implementations use kernel other than Linux, they:

- [C-2-1] MUST use a mandatory access control system that is equivalent to SELinux.

Android contains multiple defense-in-depth features that are integral to device security.

9.8. Privacy

9.8.1. Usage History

Android stores the history of the user's choices and manages such history by UsageStatsManager .

Device implementations:

- [C-0-1] MUST keep a reasonable retention period of such user history.
- [SR] Are STRONGLY RECOMMENDED to keep the 14 days retention period as configured by default in the AOSP implementation.

Android stores the system events using the StatsLog identifiers, and manages such history via the StatsManager and the IncidentManager System API.

Device implementations:

- [C-0-2] MUST only include the fields marked with DEST_AUTOMATIC in the incident report created by the System API class IncidentManager .
- [C-0-3] MUST not use the system event identifiers to log any other event than what is described in the StatsLog SDK documents. If additional system events are logged, they MAY use a different atom identifier in the range between 100,000 and 200,000.

9.8.2. Recording

Device implementations:

- [C-0-1] MUST NOT preload or distribute software components out-of-box that send the user's private information (e.g. keystrokes, text displayed on the screen, bugreport) off the device without the user's consent or clear ongoing notifications.
- [C-0-2] MUST display and obtain explicit user consent that includes exactly the same message as AOSP whenever screen casting or screen recording is enabled via MediaProjection or proprietary APIs. MUST NOT provide users an affordance to disable future display of the user consent.
- [C-0-3] MUST have an ongoing notification to the user while screen casting or screen recording is enabled. AOSP meets this requirement by showing an ongoing notification icon in the status bar.

If device implementations include functionality in the system that either captures the contents displayed on the screen and/or records the audio stream played on the device other than via the System API ContentCaptureService , or other proprietary means described in Section 9.8.6 Content Capture , they:

- [C-1-1] MUST have an ongoing notification to the user whenever this functionality is enabled and actively capturing/recording.

If device implementations include a component enabled out-of-box, capable of recording ambient audio and/or record the audio played on the device to infer useful information about user’s context, they:

- [C-2-1] MUST NOT store in persistent on-device storage or transmit off the device the recorded raw audio or any format that can be converted back into the original audio or a near facsimile, except with explicit user consent.

9.8.3. Connectivity

If device implementations have a USB port with USB peripheral mode support, they:

- [C-1-1] MUST present a user interface asking for the user's consent before allowing access to the contents of the shared storage over the USB port.

9.8.4. Network Traffic

Device implementations:

- [C-0-1] MUST preinstall the same root certificates for the system-trusted Certificate Authority (CA) store as provided in the upstream Android Open Source Project.
- [C-0-2] MUST ship with an empty user root CA store.
- [C-0-3] MUST display a warning to the user indicating the network traffic may be monitored, when a user root CA is added.

If device traffic is routed through a VPN, device implementations:

- [C-1-1] MUST display a warning to the user indicating either:
  - That network traffic may be monitored.
  - That network traffic is being routed through the specific VPN application providing the VPN.

If device implementations have a mechanism, enabled out-of-box by default, that routes network data traffic through a proxy server or VPN gateway (for example, preloading a VPN service with android.permission.CONTROL_VPN granted), they:

- [C-2-1] MUST ask for the user's consent before enabling that mechanism, unless that VPN is enabled by the Device Policy Controller via the DevicePolicyManager.setAlwaysOnVpnPackage() , in which case the user does not need to provide a separate consent, but MUST only be notified.

If device implementations implement a user affordance to toggle on the "always-on VPN" function of a 3rd-party VPN app, they:

- [C-3-1] MUST disable this user affordance for apps that do not support always-on VPN service in the AndroidManifest.xml file via setting the SERVICE_META_DATA_SUPPORTS_ALWAYS_ON attribute to false .

9.8.5. Device Identifiers

Device implementations:

- [C-0-1] MUST prevent access to the device serial number and, where applicable, IMEI/MEID, SIM serial number, and International Mobile Subscriber Identity (IMSI) from an app, unless it meets one of the following requirements:
  - is a signed carrier app that is verified by device manufacturers.
  - has been granted the READ_PRIVILEGED_PHONE_STATE permission.
  - has carrier privileges as defined in UICC Carrier Privileges .
  - is a device owner or profile owner that has been granted the READ_PHONE_STATE permission.

9.8.6. Content Capture

Android, through the System API ContentCaptureService , or by other proprietary means, supports a mechanism for device implementations to capture the following interactions between the applications and the user.

- Text and graphics rendered on-screen, including but not limited to, notifications and assist data via AssistStructure API.
- Media data, such as audio or video, recorded or played by the device.
- Input events (e.g. key, mouse, gesture, voice, video, and accessibility).
- Any other events that an application provides to the system via the Content Capture API or a similarly capable, proprietary API.
- Any text or other data sent via the TextClassifier API to the System TextClassifier i.e to the system service to understand the meaning of text, as well as generating predicted next actions based on the text.

If device implementations capture the data above, they:

- [C-0-1] MUST encrypt all such data when stored in the device. This encryption MAY be carried out using Android File Based Encryption, or any of the ciphers listed as API version 26+ described in Cipher SDK .
- [C-0-2] MUST NOT back up either raw or encrypted data using Android backup methods or any other back up methods.
- [C-0-3] MUST only send all such data and the log of the device using a privacy-preserving mechanism. The privacy-preserving mechanism is defined as “those which allow only analysis in aggregate and prevent matching of logged events or derived outcomes to individual users”, to prevent any per-user data being introspectable (e.g., implemented using a differential privacy technology such as RAPPOR).
- [C-0-4] MUST NOT associate such data with any user identity (such as Account) on the device, except with explicit user consent each time the data is associated.
- [C-0-5] MUST NOT share such data with other apps, except with explicit user consent

every time it is shared.
- [C-0-6] MUST provide user affordance to erase such data that the ContentCaptureService or the proprietary means collects if the data is stored in any form on the device.

If device implementations include a service that implements the System API ContentCaptureService , or any proprietary service that captures the data as described as above, they:

- [C-1-1] MUST NOT allow users to replace the content capture service with a user-installable application or service and MUST only allow the preinstalled service to capture such data.
- [C-1-2] MUST NOT allow any apps other than the preinstalled content capture service mechanism to be able to capture such data.
- [C-1-3] MUST provide user affordance to disable the content capture service.
- [C-1-4] MUST NOT omit user affordance to manage Android permissions that are held by the content capture service and follow Android permissions model as described in Section 9.1. Permission .
- [C-SR] Are STRONGLY RECOMMENDED to keep the content capturing service components separate, for example, not binding the service or sharing process IDs, from other system components except for the following:
  - Telephony, Contacts, System UI, and Media

9.8.7. Clipboard Access

Device implementations:

- [C-0-1] MUST NOT return a clipped data on the clipboard (e.g. via the ClipboardManager API) unless the app is the default IME or is the app that currently has focus.

9.8.8. Location

Device implementations:

- [C-0-1] MUST NOT turn on/off device location setting and Wi-Fi/Bluetooth scanning settings without explicit user consent or user initiation.
- [C-0-2] MUST provide the user affordance to access location related information including recent location requests, app level permissions and usage of Wi-Fi/Bluetooth scanning for determining location.
- [C-0-3] MUST ensure that the application using Emergency Location Bypass API [LocationRequest.setLocationSettingsIgnored()] is a user initiated emergency session (e.g. dial 911 or text to 911). For Automotive however, a vehicle MAY initiate an emergency session without active user interaction in the case a crash/accident is detected (e.g. to satisfy eCall requirements).
- [C-0-4] MUST preserve the Emergency Location Bypass API's ability to bypass device location settings without changing the settings.
- [C-0-5] MUST schedule a notification that reminds the user after an app in the background has accessed their location using the [ACCESS_BACKGROUND_LOCATION] permission.

9.8.9. Installed apps

Android apps targeting API level 30 or above cannot see details about other installed apps by default (see Package visibility in the Android SDK documentation).
Device implementations:

- [C-0-1] MUST NOT expose to any app targeting API level 30 or above details about any other installed app, unless the app is already able to see details about the other installed app through the managed APIs. This includes but is not limited to details exposed by any custom APIs added by the device implementer, or accessible via the filesystem.

9.8.10. Connectivity Bug Report

If device implementations generate bug reports using System API BUGREPORT_MODE_TELEPHONY with BugreportManager, they:

- [C-1-1] MUST obtain user consent every time the System API

BUGREPORT_MODE_TELEPHONY is called to generate a report and MUST NOT prompt the user to consent to all future requests from the application.

- [C-1-2] MUST display and obtain explicit user consent when the reports are starting to be generated and MUST NOT return the generated report to the requesting app without explicit user consent.
- [C-1-3] MUST generate requested reports containing at least the following information:
  - TelephonyDebugService dump
  - TelephonyRegistry dump
  - WifiService dump
  - ConnectivityService dump
  - A dump of the calling package's CarrierService instance (if bound)
  - Radio log buffer
- [C-1-4] MUST NOT include the following in the generated reports:
  - Any kind of information unrelated to connectivity debugging.
  - Any kind of user-installed application traffic logs or detailed profiles of user-installed applications/packages (UIDs are okay, package names are not).
- MAY include additional information that is not associated with any user identity. (e.g. vendor logs).

If device implementations include additional information (e.g vendor logs) in the bug report and that information has privacy/security/battery/storage/memory impact, they:

- [C-SR] Are STRONGLY RECOMMENDED to have a developer setting defaulted to disabled. The AOSP meets this by providing the Enable verbose vendor logging option in developer settings to include additional device-specific vendor logs in the bug reports.

9.8.11. Data blobs sharing

Android, through BlobStoreManager allows apps to contribute data blobs to the System to be shared with a selected set of apps.

If device implementations support shared data blobs as described in the SDK documentation , they:

- [C-1-1] MUST NOT share data blobs belonging to apps beyond what they intended to allow (i.e. the scope of default access and the other access modes that can be specified using BlobStoreManager.session#allowPackageAccess() , BlobStoreManager.session#allowSameSignatureAccess() , or BlobStoreManager.session#allowPublicAccess() MUST NOT be modified). The AOSP reference implementation meets these requirements.
- [C-1-2] MUST NOT send off device or share with other apps the secure hashes of data blobs (which are used to control access).

9.9. Data Storage Encryption

All devices MUST meet the requirements of section 9.9.1. Devices which launched on an API level earlier than that of this document are exempted from the requirements of sections 9.9.2 and 9.9.3; instead they MUST meet the requirements in section 9.9 of the Android Compatibility Definition document corresponding to the API level on which the device launched.

9.9.1. Direct Boot

Device implementations:

- [C-0-1] MUST implement the Direct Boot mode APIs even if they do not support Storage Encryption.
- [C-0-2] The ACTION_LOCKED_BOOT_COMPLETED and ACTION_USER_UNLOCKED Intents MUST still be broadcast to signal Direct Boot aware applications that Device Encrypted (DE) and Credential Encrypted (CE) storage locations are available for user.

9.9.2. Encryption requirements

Device implementations:

- [C-0-1] MUST encrypt the application private data (/data partition), as well as the application shared storage partition (/sdcard partition) if it is a permanent, non-removable

part of the device.

- [C-0-2] MUST enable the data storage encryption by default at the time the user has completed the out-of-box setup experience.
- [C-0-3] MUST meet the above data storage encryption requirement via implementing File Based Encryption (FBE) and Metadata Encryption .

9.9.3. Encryption Methods

If device implementations are encrypted, they:

- [C-1-1] MUST boot up without challenging the user for credentials and allow Direct Boot aware apps to access to the Device Encrypted (DE) storage after the ACTION_LOCKED_BOOT_COMPLETED message is broadcasted.
- [C-1-2] MUST only allow access to Credential Encrypted (CE) storage after the user has unlocked the device by supplying their credentials (eg. passcode, pin, pattern or fingerprint) and the ACTION_USER_UNLOCKED message is broadcasted.
- [C-1-13] MUST NOT offer any method to unlock the CE protected storage without either the user-supplied credentials, a registered escrow key or a resume on reboot implementation meeting the requirements in section 9.9.4 .
- [C-1-4] MUST use Verified Boot.
- [C-1-5] MUST encrypt file contents and filesystem metadata using AES-256-XTS or Adiantum. AES-256-XTS refers to the Advanced Encryption Standard with a 256-bit cipher key length, operated in XTS mode; the full length of the key is 512 bits. Adiantum refers to Adiantum-XChaCha12-AES, as specified at https://github.com/google/adiantum. Filesystem metadata is data such as file sizes, ownership, modes, and extended attributes (xattrs).
- [C-1-6] MUST encrypt file names using AES-256-CBC-CTS or Adiantum.
- [C-1-12] If the device has Advanced Encryption Standard (AES) instructions (such as ARMv8 Cryptography Extensions on ARM-based devices, or AES-NI on x86-based devices) then the AES-based options above for file name, file contents, and filesystem metadata encryption MUST be used, not Adiantum.
- [C-1-13] MUST use a cryptographically strong and non-reversible key derivation function (e.g. HKDF-SHA512) to derive any needed subkeys (e.g. per-file keys) from the CE and DE keys. "Cryptographically strong and non-reversible" means that the key derivation function has a security strength of at least 256 bits and behaves as a pseudorandom function family over its inputs.
- [C-1-14] MUST NOT use the same File Based Encryption (FBE) keys or subkeys for different cryptographic purposes (e.g. for both encryption and key derivation, or for two different encryption algorithms).
- The keys protecting CE and DE storage areas and filesystem metadata:
- [C-1-7] MUST be cryptographically bound to a hardware-backed Keystore. This keystore MUST be bound to Verified Boot and the device's hardware root of trust.
- [C-1-8] CE keys MUST be bound to a user's lock screen credentials.
- [C-1-9] CE keys MUST be bound to a default passcode when the user has not specified lock screen credentials.
- [C-1-10] MUST be unique and distinct, in other words no user's CE or DE key matches any other user's CE or DE keys.
- [C-1-11] MUST use the mandatorily supported ciphers, key lengths and modes.
- SHOULD make preinstalled essential apps (e.g. Alarm, Phone, Messenger) Direct Boot aware.

The upstream Android Open Source project provides a preferred implementation of File Based Encryption based on the Linux kernel "fscrypt" encryption feature, and of Metadata Encryption based on the Linux kernel "dm-default-key" feature.

9.9.4. Resume on Reboot

Resume on Reboot allows unlocking the CE storage of all apps, including those that do not yet support Direct Boot, after a reboot initiated by an OTA. This feature enables users to receive notifications from installed apps after the reboot.

An implementation of Resume-on-Reboot must continue to ensure that when a device falls into an attacker's hands, it is extremely difficult for that attacker to recover the user's CE-encrypted data, even if the device is powered on, CE storage is unlocked, and the user has unlocked the device after

receiving an OTA. For insider attack resistance, we also assume the attacker gains access to broadcast cryptographic signing keys.

Specifically:

- [C-0-1] CE storage MUST NOT be readable even for the attacker who physically has the device and then has these capabilities and limitations:
  - Can use the signing key of any vendor or company to sign arbitrary messages.
  - Can cause an OTA to be received by the device.
  - Can modify the operation of any hardware (AP, flash etc) except as detailed below, but such modification involves a delay of at least an hour and a power cycle that destroys RAM contents.
  - Cannot modify the operation of tamper-resistant hardware (eg Titan M).
  - Cannot read the RAM of the live device.
  - Cannot obtain the user's credential (PIN, pattern, password) or otherwise cause it to be entered.

By way of example, a device implementation that implements and complies with all of the descriptions found here will be compliant with [C-0-1].

9.10. Device Integrity

The following requirements ensure there is transparency to the status of the device integrity. Device implementations:

- [C-0-1] MUST correctly report through the System API method `PersistentDataBlockManager.getFlashLockState()` whether their bootloader state permits flashing of the system image. The `FLASH_LOCK_UNKNOWN` state is reserved for device implementations upgrading from an earlier version of Android where this new system API method did not exist.
- [C-0-2] MUST support Verified Boot for device integrity.

If device implementations are already launched without supporting Verified Boot on an earlier version of Android and can not add support for this feature with a system software update, they MAY be exempted from the requirement.

Verified Boot is a feature that guarantees the integrity of the device software. If device implementations support the feature, they:

- [C-1-1] MUST declare the platform feature flag `android.software.verified_boot` .
- [C-1-2] MUST perform verification on every boot sequence.
- [C-1-3] MUST start verification from an immutable hardware key that is the root of trust and go all the way up to the system partition.
- [C-1-4] MUST implement each stage of verification to check the integrity and authenticity of all the bytes in the next stage before executing the code in the next stage.
- [C-1-5] MUST use verification algorithms as strong as current recommendations from NIST for hashing algorithms (SHA-256) and public key sizes (RSA-2048).
- [C-1-6] MUST NOT allow boot to complete when system verification fails, unless the user consents to attempt booting anyway, in which case the data from any non-verified storage blocks MUST not be used.
- [C-1-7] MUST NOT allow verified partitions on the device to be modified unless the user has explicitly unlocked the bootloader.
- [C-SR] If there are multiple discrete chips in the device (e.g. radio, specialized image processor), the boot process of each of those chips is STRONGLY RECOMMENDED to verify every stage upon booting.
- [C-1-8] MUST use tamper-evident storage: for storing whether the bootloader is unlocked. Tamper-evident storage means that the bootloader can detect if the storage has been tampered with from inside Android.
- [C-1-9] MUST prompt the user, while using the device, and require physical confirmation before allowing a transition from bootloader locked mode to bootloader unlocked mode.
- [C-1-10] MUST implement rollback protection for partitions used by Android (e.g. boot, system partitions) and use tamper-evident storage for storing the metadata used for determining the minimum allowable OS version.
- [C-SR] Are STRONGLY RECOMMENDED to verify all privileged app APK files with a chain of trust rooted in partitions protected by Verified Boot.

- [C-SR] Are STRONGLY RECOMMENDED to verify any executable artifacts loaded by a privileged app from outside its APK file (such as dynamically loaded code or compiled code) before executing them or STRONGLY RECOMMENDED not to execute them at all.
- SHOULD implement rollback protection for any component with persistent firmware (e.g. modem, camera) and SHOULD use tamper-evident storage for storing the metadata used for determining the minimum allowable version.

If device implementations are already launched without supporting C-1-8 through C-1-10 on an earlier version of Android and can not add support for these requirements with a system software update, they MAY be exempted from the requirements.

The upstream Android Open Source Project provides a preferred implementation of this feature in the external/avb/ repository, which can be integrated into the bootloader used for loading Android.

Device implementations:

- [C-0-3] MUST support cryptographically verifying file content against a trusted key without reading the whole file.
- [C-0-4] MUST NOT allow the read requests on a protected file to succeed when the read content do not verify against a trusted key.
- [C-0-5] MUST enable the above-described cryptographic file verification protection for all files for the package that is installed with trusted signature files as described here .

If device implementations are already launched without the ability to verify file content against a trusted key on an earlier Android version and can not add support for this feature with a system software update, they MAY be exempted from the requirement. The upstream Android Open Source project provides a preferred implementation of this feature based on the Linux kernel fs-verity feature.

Device implementations:

- [C-R] Are RECOMMENDED to support the Android Protected Confirmation API .

If device implementations support the Android Protected Confirmation API they:

- [C-3-1] MUST report true for the ConfirmationPrompt.isSupported() API.
- [C-3-2] MUST ensure that code running in the Android OS including its kernel, malicious or otherwise, cannot generate a positive response without user interaction.
- [C-3-3] MUST ensure that the user has been able to review and approve the prompted message even in the event that the Android OS, including its kernel, is compromised.

9.11. Keys and Credentials

The Android Keystore System allows app developers to store cryptographic keys in a container and use them in cryptographic operations through the KeyChain API or the Keystore API . Device implementations:

- [C-0-1] MUST allow at least 8,192 keys to be imported or generated.
- [C-0-2] The lock screen authentication MUST rate-limit attempts and MUST have an exponential backoff algorithm. Beyond 150 failed attempts, the delay MUST be at least 24 hours per attempt.
- SHOULD not limit the number of keys that can be generated

When the device implementation supports a secure lock screen, it:

- [C-1-1] MUST back up the keystore implementation with an isolated execution environment.
- [C-1-2] MUST have implementations of RSA, AES, ECDSA and HMAC cryptographic algorithms and MD5, SHA1, and SHA-2 family hash functions to properly support the Android Keystore system's supported algorithms in an area that is securely isolated from the code running on the kernel and above. Secure isolation MUST block all potential mechanisms by which kernel or userspace code might access the internal state of the isolated environment, including DMA. The upstream Android Open Source Project (AOSP) meets this requirement by using the Trusty implementation, but another ARM TrustZone-based solution or a third-party reviewed secure implementation of a proper hypervisor-based isolation are alternative options.
- [C-1-3] MUST perform the lock screen authentication in the isolated execution

environment and only when successful, allow the authentication-bound keys to be used. Lock screen credentials MUST be stored in a way that allows only the isolated execution environment to perform lock screen authentication. The upstream Android Open Source Project provides the Gatekeeper Hardware Abstraction Layer (HAL) and Trusty, which can be used to satisfy this requirement.
- [C-1-4] MUST support key attestation where the attestation signing key is protected by secure hardware and signing is performed in secure hardware. The attestation signing keys MUST be shared across large enough number of devices to prevent the keys from being used as device identifiers. One way of meeting this requirement is to share the same attestation key unless at least 100,000 units of a given SKU are produced. If more than 100,000 units of an SKU are produced, a different key MAY be used for each 100,000 units.

Note that if a device implementation is already launched on an earlier Android version, such a device is exempted from the requirement to have a keystore backed by an isolated execution environment and support the key attestation, unless it declares the android.hardware.fingerprint feature which requires a keystore backed by an isolated execution environment.

- [C-1-5] MUST allow the user to choose the Sleep timeout for transition from the unlocked to the locked state, with a minimum allowable timeout up to 15 seconds. Automotive devices, that lock the screen whenever the head unit is turned off or the user is switched, MAY NOT have the Sleep timeout configuration.

9.11.1. Secure Lock Screen and Authentication

The AOSP implementation follows a tiered authentication model where a knowledge-factory based primary authentication can be backed by either a secondary strong biometric, or by weaker tertiary modalities.

Device implementations:

- [C-SR] Are STRONGLY RECOMMENDED to set only one of the following as the primary authentication method:
  - A numerical PIN
  - An alphanumerical password
  - A swipe pattern on a grid of exactly 3x3 dots

Note that the above authentication methods are referred as the recommended primary authentication methods in this document.

If device implementations add or modify the recommended primary authentication methods and use a new authentication method as a secure way to lock the screen, the new authentication method:

- [C-2-1] MUST be the user authentication method as described in Requiring User Authentication For Key Use .

If device implementations add or modify the authentication methods to unlock the lock screen if based on a known secret and use a new authentication method to be treated as a secure way to lock the screen:

- [C-3-1] The entropy of the shortest allowed length of inputs MUST be greater than 10 bits.
- [C-3-2] The maximum entropy of all possible inputs MUST be greater than 18 bits.
- [C-3-3] The new authentication method MUST NOT replace any of the recommended primary authentication methods (i.e. PIN, pattern, password) implemented and provided in AOSP.
- [C-3-4] The new authentication method MUST be disabled when the Device Policy Controller (DPC) application has set the password quality policy via the DevicePolicyManager.setPasswordQuality() method with a more restrictive quality constant than PASSWORD_QUALITY_SOMETHING .
- [C-3-5] New authentication methods MUST either fall back to the recommended primary authentication methods (i.e. PIN, pattern, password) once every 72 hours or less OR clearly disclose to the user that some data will not be backed up in order to preserve the privacy of their data.

If device implementations add or modify the recommended primary authentication methods to unlock the lock screen and use a new authentication method that is based on biometrics to be treated as a secure way to lock the screen, the new method:

- [C-4-1] MUST meet all requirements described in section 7.3.10 for **Class 1** (formerly **Convenience**).
- [C-4-2] MUST have a fall-back mechanism to use one of the recommended primary authentication methods which is based on a known secret.
- [C-4-3] MUST be disabled and only allow the recommended primary authentication to unlock the screen when the Device Policy Controller (DPC) application has set the keyguard feature policy by calling the method DevicePolicyManager.setKeyguardDisabledFeatures() , with any of the associated biometric flags (i.e. KEYGUARD_DISABLE_BIOMETRICS , KEYGUARD_DISABLE_FINGERPRINT , KEYGUARD_DISABLE_FACE , or KEYGUARD_DISABLE_IRIS).

If the biometric authentication methods do not meet the requirements for **Class 3** (formerly **Strong**) as described in section 7.3.10 :

- [C-5-1] The methods MUST be disabled if the Device Policy Controller (DPC) application has set the password quality policy via the DevicePolicyManager.setPasswordQuality() method with a more restrictive quality constant than PASSWORD_QUALITY_BIOMETRIC_WEAK .
- [C-5-2] The user MUST be challenged for the recommended primary authentication (eg: PIN, pattern, password) as described in [C-1-7] and [C-1-8] in section 7.3.10 .
- [C-5-3] The methods MUST NOT be treated as a secure lock screen, and MUST meet the requirements that start with C-8 in this section below.

If device implementations add or modify the authentication methods to unlock the lock screen and a new authentication method is based on a physical token or the location:

- [C-6-1] They MUST have a fall-back mechanism to use one of the recommended primary authentication methods which is based on a known secret and meet the requirements to be treated as a secure lock screen.
- [C-6-2] The new method MUST be disabled and only allow one of the recommended primary authentication methods to unlock the screen when the Device Policy Controller (DPC) application has set the policy with either the DevicePolicyManager.setKeyguardDisabledFeatures(KEYGUARD_DISABLE_TRUST_AGENTS) method or the DevicePolicyManager.setPasswordQuality() method with a more restrictive quality constant than PASSWORD_QUALITY_UNSPECIFIED .
- [C-6-3] The user MUST be challenged for one of the recommended primary authentication methods (e.g.PIN, pattern, password) at least once every 4 hours or less.
- [C-6-4] The new method MUST NOT be treated as a secure lock screen and MUST follow the constraints listed in C-8 below.

If device implementations have a secure lock screen and include one or more trust agent, which implements the TrustAgentService **System API**, they:

- [C-7-1] MUST have clear indication in the settings menu and on the lock screen when device lock is deferred or can be unlocked by trust agent(s). For example, AOSP meets this requirement by showing a text description for the "Automatically lock setting" and "Power button instantly locks" in the settings menu and a distinguishable icon on the lock screen.
- [C-7-2] MUST respect and fully implement all trust agent APIs in the DevicePolicyManager class, such as the KEYGUARD_DISABLE_TRUST_AGENTS constant.
- [C-7-3] MUST NOT fully implement the TrustAgentService.addEscrowToken() function on a device that is used as a primary personal device (e.g. handheld) but MAY fully implement the function on device implementations that are typically shared (e.g. Android Television or Automotive device).
- [C-7-4] MUST encrypt all stored tokens added by TrustAgentService.addEscrowToken() .
- [C-7-5] MUST NOT store the encryption key or escrow token on the same device where the key is used. For example, it is allowed for a key stored on a phone to unlock a user account on a TV. For Automotive devices, it is not allowed for the escrow token to be stored on any part of the vehicle.
- [C-7-6] MUST inform the user about the security implications before enabling the escrow token to decrypt the data storage.
- [C-7-7] MUST have a fall-back mechanism to use one of the recommended primary authentication methods.
- [C-7-8] The user MUST be challenged for one of the recommended primary authentication (eg: PIN, pattern, password) methods at least once every 72 hours or less.

- [C-7-9] The user MUST be challenged for one of the recommended primary authentication (eg: PIN, pattern, password) methods as described in [C-1-7] and [C-1-8] in section 7.3.10 .
- [C-7-10] MUST NOT be treated as a secure lock screen and MUST follow the constraints listed in C-8 below.
- [C-7-11] MUST NOT allow TrustAgents on primary personal devices (e.g: handheld) to unlock the device, and can only use them to keep an already unlocked device in the unlocked state for up to a maximum of 4 hours. The default implementation of TrustManagerService in AOSP meets this requirement.
- [C-7-12] MUST use a cryptographically secure (e.g UKEY2) communication channel to pass the escrow token from the storage device to the target device.

If device implementations add or modify the authentication methods to unlock the lock screen that is not a secure lock screen as described above, and use a new authentication method to unlock the keyguard:

- [C-8-1] The new method MUST be disabled when the Device Policy Controller (DPC) application has set the password quality policy via the DevicePolicyManager.setPasswordQuality() method with a more restrictive quality constant than PASSWORD_QUALITY_UNSPECIFIED .
- [C-8-2] They MUST NOT reset the password expiration timers set by DevicePolicyManager.setPasswordExpirationTimeout() .
- [C-8-3] They MUST NOT expose an API for use by third-party apps to determine the lock state.

9.11.2. StrongBox

The Android Keystore System allows app developers to store cryptographic keys in a dedicated secure processor as well as the isolated execution environment described above. Such a dedicated secure processor is called "StrongBox". Requirements C-1-3 through C-1-11 below define the requirements a device must meet to qualify as a StrongBox.

Device implementations that have a dedicated secure processor:

- [C-SR] Are STRONGLY RECOMMENDED to support StrongBox. StrongBox will likely become a requirement in a future release.

If device implementations support StrongBox, they:

- [C-1-1] MUST declare FEATURE_STRONGBOX_KEYSTORE .
- [C-1-2] MUST provide dedicated secure hardware that is used to back keystore and secure user authentication. The dedicated secure hardware may be used for other purposes as well.
- [C-1-3] MUST have a discrete CPU that shares no cache, DRAM, coprocessors or other core resources with the application processor (AP).
- [C-1-4] MUST ensure that any peripherals shared with the AP cannot alter StrongBox processing in any way, or obtain any information from the StrongBox. The AP MAY disable or block access to StrongBox.
- [C-1-5] MUST have an internal clock with reasonable accuracy (+-10%) that is immune to manipulation by the AP.
- [C-1-6] MUST have a true random number generator that produces uniformly-distributed and unpredictable output.
- [C-1-7] MUST have tamper resistance, including resistance against physical penetration, and glitching.
- [C-1-8] MUST have side-channel resistance, including resistance against leaking information via power, timing, electromagnetic radiation, and thermal radiation side channels.
- [C-1-9] MUST have secure storage which ensures confidentiality, integrity, authenticity, consistency, and freshness of the contents. The storage MUST NOT be able to be read or altered, except as permitted by the StrongBox APIs.
- To validate compliance with [C-1-3] through [C-1-9], device implementations:
  - [C-1-10] MUST include the hardware that is certified against the Secure IC Protection Profile BSI-CC-PP-0084-2014 or evaluated by a nationally

accredited testing laboratory incorporating High attack potential vulnerability assessment according to the Common Criteria Application of Attack Potential to Smartcards .
    - [C-1-11] MUST include the firmware that is evaluated by a nationally accredited testing laboratory incorporating High attack potential vulnerability assessment according to the Common Criteria Application of Attack Potential to Smartcards .
    - [C-SR] Are STRONGLY RECOMMENDED to include the hardware that is evaluated using a Security Target, Evaluation Assurance Level (EAL) 5, augmented by AVA_VAN.5. EAL 5 certification will likely become a requirement in a future release.
- [C-SR] are STRONGLY RECOMMENDED to provide insider attack resistance (IAR), which means that an insider with access to firmware signing keys cannot produce firmware that causes the StrongBox to leak secrets, to bypass functional security requirements or otherwise enable access to sensitive user data. The recommended way to implement IAR is to allow firmware updates only when the primary user password is provided via the IAuthSecret HAL. IAR will likely become a requirement in a future release.

9.11.3. Identity Credential

The Identity Credential System is defined and achieved by implementing all APIs in the android.security.identity.* package. These APIs allows app developers to store and retrieve user identity documents. Device implementations:

- [C-SR] are STRONGLY RECOMMENDED to implement the Identity Credential System.

If device implementations implement the Identity Credential System, they:

- [C-0-1] MUST return non-null for the IdentityCredentialStore#getInstance() method.
- [C-0-2] MUST implement the Identity Credential System (e.g. the android.security.identity.* APIs) with code communicating with a trusted application in an area that is securely isolated from the code running on the kernel and above. Secure isolation MUST block all potential mechanisms by which kernel or userspace code might access the internal state of the isolated environment, including DMA.
- [C-0-3] The cryptographic operations needed to implement the Identity Credential System (e.g. the android.security.identity.* APIs) MUST be performed entirely in the trusted application and private key material MUST never leave the isolated execution environment unless specifically required by higher-level APIs (e.g. the createEphemeralKeyPair() method).
- [C-0-4] The trusted application MUST be implemented in a way such that its security properties are not affected (e.g. credential data is not released unless access control conditions are satisfied, MACs can't be produced for arbitrary data) even if Android is misbehaving or compromised.

9.12. Data Deletion

All device implementations:

- [C-0-1] MUST provide users a mechanism to perform a "Factory Data Reset".
- [C-0-2] MUST delete all data on the userdata filesystem.
- [C-0-3] MUST delete the data in such a way that will satisfy relevant industry standards such as NIST SP800-88.
- [C-0-4] MUST trigger the above "Factory Data Reset" process when the DevicePolicyManager.wipeData() API is called by the primary user's Device Policy Controller app.
- MAY provide a fast data wipe option that conducts only a logical data erase.

9.13. Safe Boot Mode

Android provides Safe Boot Mode, which allows users to boot up into a mode where only preinstalled system apps are allowed to run and all third-party apps are disabled. This mode, known as "Safe Boot Mode", provides the user the capability to uninstall potentially harmful third-party apps.

Device implementations are:

- [SR] STRONGLY RECOMMENDED to implement Safe Boot Mode.

If device implementations implement Safe Boot Mode, they:

- [C-1-1] MUST provide the user an option to enter Safe Boot Mode in such a way that is uninterruptible from third-party apps installed on the device, except when the third-party app is a Device Policy Controller and has set the UserManager.DISALLOW_SAFE_BOOT flag as true.
- [C-1-2] MUST provide the user the capability to uninstall any third-party apps within Safe Mode.
- SHOULD provide the user an option to enter Safe Boot Mode from the boot menu using a workflow that is different from that of a normal boot.

9.14. Automotive Vehicle System Isolation

Android Automotive devices are expected to exchange data with critical vehicle subsystems by using the vehicle HAL to send and receive messages over vehicle networks such as CAN bus.

The data exchange can be secured by implementing security features below the Android framework layers to prevent malicious or unintentional interaction with these subsystems.

9.15. Subscription Plans

"Subscription plans" refer to the billing relationship plan details provided by a mobile carrier through SubscriptionManager.setSubscriptionPlans() .

All device implementations:

- [C-0-1] MUST return subscription plans only to the mobile carrier app that has originally provided them.
- [C-0-2] MUST NOT remotely back up or upload subscription plans.
- [C-0-3] MUST only allow overrides, such as SubscriptionManager.setSubscriptionOverrideCongested() , from the mobile carrier app currently providing valid subscription plans.

9.16. Application Data Migration

If device implementations include a capability to migrate data from a device to another device and do not limit the application data it copies to what is configured by the application developer in the manifest via android:fullBackupContent attribute, they:

- [C-1-1] MUST NOT initiate transfers of application data from devices on which the user has not set a primary authentication as described in 9.11.1 Secure Lock Screen and Authentication .
- [C-1-2] MUST securely confirm the primary authentication on the source device and confirm with the user intent to copy the data on the source device before any data is transferred.
- [C-1-3] MUST use security key attestation to ensure that both the source device and the target device in the device-to-device migration are legitimate Android devices and have a locked bootloader.
- [C-1-4] MUST only migrate application data to the same application on the target device, with the same package name AND signing certificate.
- [C-1-5] MUST show an indication that the source device has had data migrated by a device-to-device data migration in the settings menu. A user SHOULD NOT be able to remove this indication.

10. Software Compatibility Testing

Device implementations MUST pass all tests described in this section. However, note that no software test package is fully comprehensive. For this reason, device implementers are **STRONGLY RECOMMENDED** to make the minimum number of changes as possible to the reference and preferred implementation of Android available from the Android Open Source Project. This will minimize the risk of introducing bugs that create incompatibilities requiring rework and potential device updates.

10.1. Compatibility Test Suite

Device implementations:

- [C-0-1] MUST pass the Android Compatibility Test Suite (CTS) available from the Android Open Source Project, using the final shipping software on the device.
- [C-0-2] MUST ensure compatibility in cases of ambiguity in CTS and for any reimplementations of parts of the reference source code.

The CTS is designed to be run on an actual device. Like any software, the CTS may itself contain bugs. The CTS will be versioned independently of this Compatibility Definition, and multiple revisions of the CTS may be released for Android 11.

Device implementations:

- [C-0-3] MUST pass the latest CTS version available at the time the device software is completed.
- SHOULD use the reference implementation in the Android Open Source tree as much as possible.

10.2. CTS Verifier

The CTS Verifier is included with the Compatibility Test Suite, and is intended to be run by a human operator to test functionality that cannot be tested by an automated system, such as correct functioning of a camera and sensors.

Device implementations:

- [C-0-1] MUST correctly execute all applicable cases in the CTS verifier.

The CTS Verifier has tests for many kinds of hardware, including some hardware that is optional.

Device implementations:

- [C-0-2] MUST pass all tests for hardware that they possess; for instance, if a device possesses an accelerometer, it MUST correctly execute the Accelerometer test case in the CTS Verifier.

Test cases for features noted as optional by this Compatibility Definition Document MAY be skipped or omitted.

- [C-0-2] Every device and every build MUST correctly run the CTS Verifier, as noted above. However, since many builds are very similar, device implementers are not expected to explicitly run the CTS Verifier on builds that differ only in trivial ways. Specifically, device implementations that differ from an implementation that has passed the CTS Verifier only by the set of included locales, branding, etc. MAY omit the CTS Verifier test.

11. Updatable Software

- [C-0-1] Device implementations MUST include a mechanism to replace the entirety of the system software. The mechanism need not perform “live” upgrades—that is, a device restart MAY be required. Any method can be used, provided that it can replace the entirety of the software preinstalled on the device. For instance, any of the following approaches will satisfy this requirement:
  - “Over-the-air (OTA)” downloads with offline update via reboot.
  - “Tethered” updates over USB from a host PC.
  - “Offline” updates via a reboot and update from a file on removable storage.
- [C-0-2] The update mechanism used MUST support updates without wiping user data. That is, the update mechanism MUST preserve application private data and application shared data. Note that the upstream Android software includes an update mechanism that satisfies this requirement.
- [C-0-3] The entire update MUST be signed and the on-device update mechanism MUST verify the update and signature against a public key stored on device.
- [C-SR] The signing mechanism is STRONGLY RECOMMENDED to hash the update with SHA-256 and validate the hash against the public key using ECDSA NIST P-256.

If the device implementations includes support for an unmetered data connection such as 802.11 or Bluetooth PAN (Personal Area Network) profile, then, they:

- [C-1-1] MUST support OTA downloads with offline update via reboot.

For device implementations that are launching with Android 6.0 and later, the update mechanism SHOULD support verifying that the system image is binary identical to expected result following an OTA. The block-based OTA implementation in the upstream Android Open Source Project, added since Android 5.1, satisfies this requirement.

Also, device implementations SHOULD support A/B system updates . The AOSP implements this feature using the boot control HAL.

If an error is found in a device implementation after it has been released but within its reasonable product lifetime that is determined in consultation with the Android Compatibility Team to affect the compatibility of third-party applications, then:

- [C-2-1] The device implementer MUST correct the error via a software update available that can be applied per the mechanism just described.

Android includes features that allow the Device Owner app (if present) to control the installation of system updates. If the system update subsystem for devices report android.software.device_admin then, they:

- [C-3-1] MUST implement the behavior described in the SystemUpdatePolicy class.

12. Document Changelog

For a summary of changes to the Compatibility Definition in this release:

- Document changelog

For a summary of changes to individuals sections:

1. Introduction
2. Device Types
3. Software
4. Application Packaging
5. Multimedia
6. Developer Tools and Options
7. Hardware Compatibility
8. Performance and Power
9. Security Model
10. Software Compatibility Testing
11. Updatable Software
12. Document Changelog
13. Contact Us

12.1. Changelog Viewing Tips

Changes are marked as follows:

- **CDD**
  Substantive changes to the compatibility requirements.
- **Docs**
  Cosmetic or build related changes.

For best viewing, append the `pretty=full` and `no-merges` URL parameters to your changelog URLs.

13. Contact Us

You can join the android-compatibility forum and ask for clarifications or bring up any issues that you think the document does not cover.