android

Compatibility Definition

Android 8.0

Last updated: September 1, 2017

Table of Contents

1. Introduction

This document enumerates the requirements that must be met in order for devices to be compatible with Android 8.0.

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 8.0. A “device implementation” or “implementation is the hardware/software solution so developed.

To be considered compatible with Android 8.0, 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 the MUST and STRONGLY RECOMMENDED 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 on 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
- Condition ID
    - When the requirement is unconditional, this ID is set as 0.
    - When the requirement is conditional, 1 is assinged 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.

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 2.5 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

Screen Size (Section 7.1.1.1)

Handheld device implementations:

- [H-0-1] MUST have a screen at least 2.5 inches in physical diagonal size. *

Screen Density (Section 7.1.1.3)

Handheld device implementations:

- [H-SR] Are STRONGLY RECOMMENDED to provide users an affordance to change the display size.

Legacy Application Compatibility Mode (Section 7.1.5)

Handheld device implementations:

- [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.

Keyboard (Section 7.2.1)

Handheld device implementations:

- [H-0-1] MUST include support for third-party Input Method Editor (IME) applications.

Navigation Keys (Section 7.2.3)

Handheld device implementations:

- [H-0-1] MUST provide the Home, Recents, and Back functions.
- [H-0-2] MUST send both the normal and long press event of the the Back function (`KEYCODE_BACK`) to the foreground application.

Touchscreen Input (Section 7.2.4)

Handheld device implementations:

- [H-0-1] MUST support touchscreen input.

Accelerometer (Section 7.3.1)

Handheld device implementations:

- [H-SR] Are STRONGLY RECOMMENDED to include a 3-axis accelerometer.

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

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

Gyroscope (Section 7.3.4)

If Handheld device implementations include a gyroscope, they:

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

Proximity Sensor (Section 7.3.8)

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

- SHOULD include a proximity sensor.

Pose Sensor (Section 7.3.12)

Handheld device implementations:

- Are RECOMMENDED to support pose sensor with 6 degrees of freedom.

Bluetooth (Section 7.4.3)
Handheld device implementations:

- SHOULD include support for Bluetooth and Bluetooth LE.

Data Saver (Section 7.4.7)
If Handheld device implementations include a metered connection, they:

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

Minimum Memory and Storage (Section 7.6.1)
Handheld device implementations:

- [H-0-1] MUST have at least 4GB of non-volatile storage available for application private data (a.k.a. "/data" partition)
- [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 are 32-bit:

- [H-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
- [H-2-1] 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
- [H-3-1] 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
- [H-4-1] 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 Handheld device implementations are 64-bit:

- [H-5-1] 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
- [H-6-1] 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
- [H-7-1] 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
- [H-8-1] 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.

Application Shared Storage (Section 7.6.2)

Handheld device implementations:

- [H-0-1] MUST NOT provide an application shared storage smaller than 1 GiB.

USB peripheral mode (Section 7.7.1)

Handheld device implementations:

- SHOULD include a USB port supporting peripheral mode.

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

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

Microphone (Section 7.8.1)

Handheld device implementations:

- [H-0-1] MUST include a microphone.

Audio Output (Section 7.8.2)

Handheld device implementations:

- [H-0-1] MUST have an audio output and declare `android.hardware.audio.output` .

Virtual Reality Mode (Section 7.9.1)

If Handheld device implementations include support for the VR mode, they:

- [H-1-1] MUST declare the `android.software.vr.mode` feature. *

If device implementations declare `android.software.vr.mode` feature, they:

- [H-2-1] MUST include an application implementing `android.service.vr.VrListenerService` that can be enabled by VR applications via `android.app.Activity#setVrModeEnabled` . *

Virtual Reality High Performance (Section 7.9.2)

If Handheld device implementations are capable of meeting all the requirements to declare the `android.hardware.vr.high_performance` feature flag, they:

- [H-1-1] MUST declare the `android.hardware.vr.high_performance` feature flag. *

2.2.2. Multimedia

Audio Encoding (Section 5.1.1)

Handheld device implementations MUST support the following audio encoding:

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

Audio Decoding (Section 5.1.2)

Handheld device implementations MUST support the following audio decoding:

- [H-0-1] AMR-NB
- [H-0-2] AMR-WB

Video Encoding (Section 5.2)

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

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

Video Decoding (Section 5.3)

Handheld device implementations MUST support the following video decoding:

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

2.2.3. Software

WebView Compatibility (Section 3.4.1)

Handheld device implementations:

- [H-0-1] MUST provide a complete implementation of the `android.webkit.Webview` API.

Browser Compatibility (Section 3.4.2)

Handheld device implementations:

- [H-0-1] MUST include a standalone Browser application for general user web browsing.

Launcher (Section 3.8.1)

Handheld device implementations:

- [H-SR] Are STRONGLY RECOMMENDED to implement a default launcher that supports in-app pinning of shortcuts and widgets.
- [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.
- [H-SR] Are STRONGLY RECOMMENDED to include a default launcher app that shows badges for the app icons.

Widgets (Section 3.8.2)

Handheld device implementations:

- [H-SR] Are STRONGLY RECOMMENDED to support third-party app widgets.

Notifications (Section 3.8.3)

Handheld device implementations:

- [H-0-1] MUST allow third-party apps to notify users of notable events through the `Notification` and `NotificationManager` API classes.
- [H-0-2] MUST support rich notifications.
- [H-0-3] MUST support heads-up notifications.
- [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.

Search (Section 3.8.4)

Handheld device implementations:

- [H-SR] Are STRONGLY RECOMMENDED to implement an assistant on the device to handle the Assist action .

Lock Screen Media Control (Section 3.8.10)

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

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

Device administration (Section 3.9)

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

- [H-1-1] MUST implement the full range of device administration policies defined in the Android SDK documentation.

Accessibility (Section 3.10)

Handheld device implementations:

- [H-SR] MUST support third-party accessibility services.
- [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 preloaded Text-to-speech engine) accessibility services as provided in the talkback open source project .

Text-to-Speech (Section 3.11)

Handheld device implementations:

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

Quick Settings (Section 3.13)

Handheld device implementations:

- [H-SR] Are STRONGLY RECOMMENDED to include a Quick Settings UI component.

Companion Device Pairing (Section 3.15)

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

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

2.2.4. Performance and Power

User Experience Consistency (Section 8.1)

For handheld device implementations:

- [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.
- [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.
- [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.

File I/O Access Performance (Section 8.2)

Handheld device implementations:

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

Power-Saving Modes (Section 8.3)

For handheld device implementations:

- [H-0-1] All Apps exempted from App Standby and Doze power-saving modes MUST be made visible to the end user.

- [H-0-2] The triggering, maintenance, wakeup algorithms and the use of global system settings of App Standby and Doze power-saving modes MUST not deviate from the Android Open Source Project.

Power Consumption Accounting (Sections 8.4)

Handheld device implementations:

- [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.
- [H-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [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.
- [H-0-4] MUST 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.

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

- [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

Permissions (Sections 9.1)

Handheld device implementations:

- [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.

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

Non-touch Navigation (Section 7.2.2)

Television device implementations:

- [T-0-1] MUST support D-pad .

Navigation Keys (Section 7.2.3)
Television device implementations:

- [T-0-1] MUST provide the Home and Back functions.
- [T-0-2] MUST send both the normal and long press event of the the Back function (`KEYCODE_BACK`) to the foreground application.

Button Mappings (Section 7.2.6.1)
Television device implementations:

- [T-0-1] MUST include support for game controllers and declare the `android.hardware.gamepad` feature flag.

Remote Control (Section 7.2.7)
Television device implementations:

- SHOULD provide a remote control from which users can access non-touch navigation and core navigation keys inputs.

Gyroscope (Section 7.3.4)
If Television device implementations include a gyroscope, they:

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

Bluetooth (Section 7.4.3)
Television device implementations:

- [T-0-1] MUST support Bluetooth and Bluetooth LE.

Minimum Memory and Storage (Section 7.6.1)
Television device implementations:

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

Microphone (Section 7.8.1)
Television device implementations:

- SHOULD include a microphone.

Audio Output (Section 7.8.2)
Television device implementations:

- [T-0-1] MUST have an audio output and declare `android.hardware.audio.output` .

2.3.2. Multimedia

Audio Encoding (Section 5.1)

Television device implementations MUST support the following audio encoding:

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

Video Encoding (Section 5.2)
Television device implementations MUST support the following video encoding:

- [T-0-1] H.264 AVC
- [T-0-2] VP8

H-264 (Section 5.2.2)
Television device implementations are:

- [T-SR] STRONGLY RECOMMENDED to support H.264 encoding of 720p and 1080p resolution videos.
- [T-SR] STRONGLY RECOMMENDED to support H.264 encoding of 1080p resolution video at 30 frame-per-second (fps).

Video Decoding (Section 5.3)
Television device implementations MUST support the following video decoding:

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

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

- [T-SR] MPEG-2

H.264 (Section 5.3.4)
If Television device implementations support H.264 decoders, they:

- [T-1-1] MUST support High Profile Level 4.2 and the HD 1080p (at 60 fps) decoding profile.
- [T-1-2] MUST be capable of decoding videos with both HD profiles as indicated in the following table and encoded with either the Baseline Profile, Main Profile, or the High Profile Level 4.2

H.265 (HEVC) (Section 5.3.5)
If Television device implementations support H.265 codec and the HD 1080p decoding profile, they:

- [T-1-1] MUST support the Main Profile Level 4.1 Main tier.
- [T-SR] Are STRONGLY RECOMMENDED to support 60 fps video frame rate for HD 1080p.

If Television device implementations support H.265 codec and the UHD decoding profile, then:

* [T-2-1] The codec MUST support Main10 Level 5 Main Tier profile.

VP8 (Section 5.3.6)
If Television device implementations support VP8 codec, they:

* [T-1-1] MUST support the HD 1080p60 decoding profile.

If Television device implementations support VP8 codec and support 720p, they:

* [T-2-1] MUST support the HD 720p60 decoding profile.

VP9 (Section 5.3.7)
If Television device implementations support VP9 codec and the UHD video decoding, they:

* [T-1-1] MUST support 8-bit color depth and SHOULD support VP9 Profile 2 (10-bit).

If Television device implementations support VP9 codec, the 1080p profile and VP9 hardware decoding, they:

* [T-2-1] MUST support 60 fps for 1080p.

Secure Media (Section 5.8)
If device implementations are Android Television devices and support 4K resolution, they:

* [T-1-1] MUST support HDCP 2.2 for all wired external displays.

If Television device implementations don't support 4K resolution, they:

* [T-2-1] MUST support HDCP 1.4 for all wired external displays.

Television device implementations:

* [T-SR] Are STRONGLY RECOMMENDED to support simulataneous decoding of secure streams. At minimum, simultaneous decoding of two steams is STRONGLY RECOMMENDED.

Audio Output Volume (Section 5.5.3)
Television device implementations:

* [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).

2.3.3. Software

Television device implementations:

* [T-0-1] MUST declare the features `android.software.leanback` and `android.hardware.type.television`.

WebView compatibility (Section 3.4.1)
Television device implementations:

* [T-0-1] MUST provide a complete implementation of the `android.webkit.Webview` API.

Lock Screen Media Control (Section 3.8.10)

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

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

Multi-windows (Section 3.8.14)

Television device implementations:

* [T-SR] Are STRONGLY RECOMMENDED to support picture-in-picture (PIP) mode multi-window.

Accessibility (Section 3.10)

Television device implementations:

* [T-SR] MUST support third-party accessibility services.
* [T-SR] Android Television device implementations 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 preloaded Text-to-speech engine) accessibility services as provided in the talkback open source project .

Text-to-Speech (Section 3.11)

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

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

TV Input Framework (Section 3.12)

Television device implementations:

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

2.3.4. Performance and Power

User Experience Consistency (Section 8.1)

For Television device implementations:

* [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.

File I/O Access Performance (Section 8.2)

Television device implementations:

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

- [T-0-4] MUST ensure a random read performance of at least 3.5MB/s.

Power-Saving Modes (Section 8.3)

For Television device implementations:

- [T-0-1] All Apps exempted from App Standby and Doze power-saving modes MUST be made visible to the end user.
- [T-0-2] The triggering, maintenance, wakeup algorithms and the use of global system settings of App Standby and Doze power-saving modes MUST not deviate from the Android Open Source Project.

Power Consumption Accounting (Sections 8.4)

Television device implementations:

- [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.
- [T-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [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.
- SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.
- [T-0-4] MUST make this power usage available via the `adb shell dumpsys batterystats` shell command to the app developer.

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

Screen Size (Section 7.1.1.1)

Watch device implementations:

- [W-0-1] MUST have a screen with the physical diagonal size in the range from 1.1 to 2.5 inches.

Navigation Keys (Section 7.2.3)

Watch device implementations:

- [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` .

Touchscreen Input (Section 7.2.4)
Watch device implementations:

- [W-0-2] MUST support touchscreen input.

Accelerometer (Section 7.3.1)
Watch device implementations:

- [W-SR] Are STRONGLY RECOMMENDED to include a 3-axis accelerometer.

Bluetooth (Section 7.4.3)
Watch device implementations:

- [W-0-1] MUST support Bluetooth.

Minimum Memory and Storage (Section 7.6.1)
Watch device implementations:

- [W-0-1] MUST have at least 1GB of non-volatile storage available for application private data (a.k.a. "/data" partition)
- [W-0-2] MUST have at least 416MB memory available to the kernel and userspace.

Microphone (Section 7.8.1)
Watch device implementations:

- [W-0-1] MUST include a microphone.

Audio Output (Section 7.8.1)
Watch device implementations:

- MAY but SHOULD NOT have audio output.

2.4.2. Multimedia

No additional requirements.

2.4.3. Software

Watch device implementations:

- [W-0-1] MUST declare the feature android.hardware.type.watch.
- [W-0-2] MUST support uiMode = UI_MODE_TYPE_WATCH .

Search (Section 3.8.4)
Watch device implementations:

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

Accessibility (Section 3.10)
Watch device implementations that declare the `android.hardware.audio.output` feature flag:

- [W-1-1] MUST support third-party accessibility services.
- [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 preloaded Text-to-speech engine) accessibility services as provided in the [talkback open source project](#) .

Text-to-Speech (Section 3.11)

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

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

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

Screen Size (Section 7.1.1.1)

Automotive device implementations:

- [A-0-1] MUST have a screen at least 6 inches in physical diagonal size.
- [A-0-2] MUST have a screen size layout of at least 750 dp x 480 dp.

Navigation Keys (Section 7.2.3)

Automotive device implementations:

- [A-0-1] MUST provide the Home function and MAY provide Back and Recent functions.
- [A-0-2] MUST send both the normal and long press event of the the Back function (`KEYCODE_BACK`) to the foreground application.

Accelerometer (Section 7.3.1)

Automotive device implementations:

- [A-SR] Are STRONGLY RECOMMENDED to include a 3-axis accelerometer.

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

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

GPS (Section 7.3.3)

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

- [A-1-1] GNSS technology generation MUST be the year "2017" or newer.

Gyroscope (Section 7.3.4)

If Automotive device implementations include a gyroscope, they:

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

Android Automotive-only sensors (Section 7.3.11) Current Gear (Section 7.3.11.1)

Automotive device implementations:

- SHOULD provide current gear as `SENSOR_TYPE_GEAR` .

Day Night Mode (Section 7.3.11.2)

Automotive device implementations:

- [A-0-1] MUST support day/night mode defined as `SENSOR_TYPE_NIGHT` .
- [A-0-2] The value of the `SENSOR_TYPE_NIGHT` 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 .

Driving Status (Section 7.3.11.3)

Automotive device implementations:

- [A-0-1] MUST support driving status defined as `SENSOR_TYPE_DRIVING_STATUS` , with a default value of `DRIVE_STATUS_UNRESTRICTED` when the vehicle is fully stopped and parked. It is the responsibility of device manufacturers to configure `SENSOR_TYPE_DRIVING_STATUS` in compliance with all laws and regulations that apply to markets where the product is shipping.

Wheel Speed (Section 7.3.11.4)

Automotive device implementations:

- [A-0-1] MUST provide vehicle speed defined as `SENSOR_TYPE_CAR_SPEED` .

Bluetooth (Section 7.4.3)

Automotive device implementations:

- [A-0-1] MUST support Bluetooth and SHOULD support Bluetooth LE.
- [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).
- SHOULD support Message Access Profile (MAP).

Minimum Network Capability (Section 7.4.5)

Automotive device implementations:

- SHOULD include support for cellular network based data connectivity.

Minimum Memory and Storage (Section 7.6.1)
Automotive device implementations:

- [A-0-1] MUST have at least 4GB of non-volatile storage available for application private data (a.k.a. "/data" partition)

USB peripheral mode (Section 7.7.1)
Automotive device implementations:

- SHOULD include a USB port supporting peripheral mode.

Microphone (Section 7.8.1)
Automotive device implementations:

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

Audio Output (Section 7.8.2)
Automotive device implementations:

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

2.5.2. Multimedia

Audio Encoding (Section 5.1)
Automotive device implementations MUST support the following audio encoding:

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

Video Encoding (Section 5.2)
Automotive device implementations MUST support the following video encoding:

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

Video Decoding (Section 5.3)
Automotive device implementations MUST support the following video decoding:

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

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

- [A-SR] H.265 HEVC

2.5.3. Software

Automotive device implementations:

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

WebView Compatibility (Section 3.4.1)

Automotive device implementations:

- [A-0-1] MUST provide a complete implementation of the `android.webkit.Webview API` .

Notifications (Section 3.8.3)

Android Automotive device implementations:

- [A-0-1] MUST display notifications that use the `Notification.CarExtender` API when requested by third-party applications.

Search (Section 3.8.4)

Automotive device implementations:

- [A-0-1] MUST implement an assistant on the device to handle the Assist action .

Media UI (Section 3.14)

Automotive device implementations:

- [A-0-1] MUST include a UI framework to support third-party apps using the media APIs as described in section 3.14.

2.5.4. Performance and Power

Power-Saving Modes (Section 8.3)

For Automotive device implementations:

- [A-0-1] All Apps exempted from App Standby and Doze power-saving modes MUST be made visible to the end user.
- [A-0-2] The triggering, maintenance, wakeup algorithms and the use of global system settings of App Standby and Doze power-saving modes MUST not deviate from the Android Open Source Project.

Power Consumption Accounting (Sections 8.4)

Automotive device implementations:

- [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.
- [A-0-2] MUST report all power consumption values in milliampere hours (mAh).
- [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.

- SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.
- [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

Multi-User Support (Section 9.5)

If Automotive device implementations include multiple users, they:

- [A-1-1] MUST include a guest account that allows all functions provided by the vehicle system without requiring a user to log in.

Automotive Vehicle System Isolation (Section 9.14)

Automotive device implementations:

- [A-0-1] MUST gatekeep messages from Android framework vehicle subsystems, e.g., whitelisting permitted message types and message sources.
- [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.

2.6. Tablet Requirements

An **Android Tablet device** refers to an Android device implementation that is typically used by holding in both hands and not in a clamshell form-factor.

Android device implementations are classified as a Tablet 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 7 to 18 inches.

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

2.6.1. Hardware

Screen Size (Section 7.1.1.1)

Tablet device implementations:

- [Ta-0-1] MUST have a screen in the range of 7 to 18 inches.

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

- 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.

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.

- [C-0-1] Device implementations 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] Device implementations MUST support/preserve all classes, methods, and associated elements marked by the TestApi annotation (@TestApi).
- [C-0-3] Device implementations 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] Device implementations 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.

3.1.1. Android Extensions

Android includes the support of extending the managed APIs while keeping the same API level version.

- [C-0-1] Android device implementations MUST preload the AOSP implementation of both the shared library `ExtShared` and services `ExtServices` with versions higher 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.

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 8.0 .
VERSION.SDK	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 8.0, this field MUST have the integer value 8.0_INT.
VERSION.SDK_INT	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 8.0, this field MUST have the integer value 8.0_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. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
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.
	A string that uniquely identifies this build. It SHOULD be reasonably

FINGERPRINT	human-readable. It MUST follow this template: **$(BRAND)/$(PRODUCT)/ $(DEVICE):$(VERSION.RELEASE)/$(ID)/$(VERSION.INCREMENTAL):$(TYPE)/$(TAGS)** For example: **acme/myproduct/ mydevice:8.0/LMYXX/3359:userdebug/test-keys** The fingerprint MUST NOT include whitespace characters. If other fields included in the template above have whitespace characters, they MUST be replaced in the build fingerprint with another character, such as the underscore ("_") character. 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_-]+$”. This product name MUST NOT change during the lifetime of the product.
SERIAL	A hardware serial number, which MUST be available and unique across devices with the same MODEL and MANUFACTURER. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^([a-zA-Z0-9]{6,20})$”.
TAGS	A comma-separated list of tags chosen by the device implementer that further distinguishes the build. This field MUST have one of the values corresponding to the three typical Android platform signing configurations: release-keys, dev-keys, 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._-]+$”.
getRadioVersion()	MUST (be or return) a value chosen by the device implementer identifying the specific internal radio/modem version used in the device, in human-readable format. If a device does not have any internal radio/modem it MUST return NULL. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression “^[a-zA-Z0-9._-,]+$”.

3.2.3. Intent Compatibility

3.2.3.1. Core Application Intents

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

- [C-0-1] Device implementations MUST include these application, service components, or at least a handler, for all the public intent filter patterns defined by the the following core Android applications in AOSP:
    - Desk Clock
    - Browser
    - Calendar
    - Contacts
    - Gallery
    - GlobalSearch
    - Launcher
    - Music
    - Settings

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 to be overridden by third-party applications. The upstream Android open source implementation allows this by default.
- [C-0-2] Dvice 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 UIR intent filters as default app handlers for their UIRs.
- 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 used by

the core apps 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 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.

3.2.3.5. Default App Settings

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.

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.

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

- [C-4-1] MUST honor the android.telecom.action.CHANGE_DEFAULT_DIALER intent to show a dialog to allow the user to change the default Phone application.

If device implementations support the VoiceInteractionService, they:

- [C-5-1] MUST honor the android.settings.ACTION_VOICE_INPUT_SETTINGS intent to show a default app settings menu for voice input and assist.

3.2.4. Activities on secondary displays

If device implementations allow launching normal Android Activities on secondary displays, 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 destory all activities, when a display with the `Display.FLAG_PRIVATE` flag is removed.
- [C-1-5] MUST resize accordingly all activities on a `VirtualDisplay` if the display itself is resized.
- MAY show an IME (input method editor, a user control that enables users to enter text) on the primary display, when a text input field becomes focused on a secondary display.
- SHOULD implement the input focus on the secondary display independently of the primary display, when touch or key inputs are supported.
- 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 primary and secondary displays have different android.util.DisplayMetrics :

- [C-2-1] Non-resizeable activities (that have `resizeableActivity=false` in `AndroidManifest.xml`) and apps targeting API level 23 or lower MUST NOT be allowed on secondary displays.

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

Device implementers are:

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-0-4] MUST support the equivalent 32-bit ABI if any 64-bit ABI is supported.
- [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, only those ABIs documented and described in the latest version of the Android NDK ABI Management documentation , and MUST include support for the Advanced SIMD (a.k.a. NEON) extension.
- [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)
    - 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)
    - 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 symobls, 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 the Android NDK may introduce support for additional ABIs.

3.3.2. 32-bit ARM Native Code Compatibility

If device implementations are 64-bit ARM devices, then:

- [C-1-1] Although the ARMv8 architecture deprecates several CPU operations, including some operations used in existing native code, the following deprecated operations MUST remain available to 32-bit native ARM code, either through native CPU support or through software emulation:
  - SWP and SWPB instructions
  - SETEND instruction
  - CP15ISB, CP15DSB, and CP15DMB barrier operations

If device implementations include a 32-bit ARM ABI, they:

- [C-2-1] MUST include the following lines in `/proc/cpuinfo` when it is read by 32-bit ARM applications to ensure compatibility with applications built using legacy versions of Android NDK.
  - `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).
- SHOULD not alter `/proc/cpuinfo` when read by 64-bit ARM or non-ARM applications.

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 8.0 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 value of the $(MODEL) string MUST be the same as the value for android.os.Build.MODEL.
  - The value of 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 .

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

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.

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.

3.6. 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.*`
- `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
	160 dpi (mdpi)	
	213 dpi (tvdpi)	
	240 dpi (hdpi)	36MB
	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
small/normal	120 dpi (ldpi)	32MB
	160 dpi (mdpi)	
	213 dpi (tvdpi)	48MB
	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
	160 dpi (mdpi)	48MB
	213 dpi (tvdpi)	80MB
	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
	160 dpi (mdpi)	80MB
	213 dpi (tvdpi)	96MB
	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 and widgets, they:

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

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

- [C-3-1] MUST report `false` for `ShortcutManager.isRequestPinShortcutSupported()` and `AppWidgetManager.html#isRequestPinAppWidgetSupported()` .

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 Design Guidelines in the Android SDK documentation for details.
- MAY support application widgets on the lock screen.

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.
- SHOULD support rich notifications.
- SHOULD present some higher priority notifications as heads-up notifications.
- SHOULD have 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.

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 impelementations 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.

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 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.
- [C-1-2] 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.
- [SR] STRONGLY RECOMMENDED to use long press on `HOME` key as this designated interaction.

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.

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 20 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] Device implementations 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.
- [C-1-2] 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 declare the `android.software.autofill` feature flag, they:

- [C-2-1] 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.

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)

Android includes support for interactivescreensavers , 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. Android Watch devices MAY implement screen savers, but other types of device implementations SHOULD include support for screen savers and provide a settings option for users toconfigure screen savers in response to the `android.settings.DREAM_SETTINGS` intent.

3.8.12. Location

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

- [C-1-1] location modes MUST be displayed 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.

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] Applications can indicate whether they are capable of operating in multi-window mode in the `AndroidManifest.xml` file, either explicitly via setting the `android:resizeableActivity` attribute to `true` or implicitly by having the targetSdkVersion > 24. Apps that explicitly set this attribute to `false` in their manifest MUST NOT be launched in multi-window mode. Older apps with targetSdkVersion < 24 that did not set this `android:resizeableActivity` attribute MAY be launched in multi-window mode, but the system MUST provide warning that the app may not work as expected in multi-window mode.
- [C-1-3] MUST NOT offer split-screen or freeform mode if the screen height < 440 dp and the the screen width < 440 dp.
- 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 minimum width and height of 108 dp for the PIP window and minimum width of 240 dp and height of 135 dp for the PIP window when the `Configuration.uiMode` is configured as `UI_MODE_TYPE_TELEVISION`

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 .
- [C-1-3] MUST declare the support of manged profiles via the `android.software.managed_users` feature flag, except for when the device is configured so that it would report itself as a low RAM device or so that it allocate internal (non-removable) storage as shared storage.

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 a Device 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_DEV` .
    - [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_DEV` .
    - [C-1-7] MUST not enroll any DPC application as the Device Owner App any more.
- [C-1-2] MUST NOT set an application (including pre-installed app) as the Device Owner app without explicit consent or action from the user or the administrator of the device.

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.
- [C-1-10] MUST support the ability to specify a separate lock screen meeting the following

requirements to grant access to apps running in a managed profile.
    - 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.

3.10. 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-1-3] 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.
- [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 preloaded accessibility services, they:

- [C-2-1] MUST implement these preloaded accessibility services as [Direct Boot aware](https://developer.android.com/reference/android/content/pm/ComponentInfo.html#directBootAware) 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 preload a TV application (TV App) and meet all requirements described in section 3.12.1 .

3.12.1. TV App

If device implementations support TIF:

- [C-1-1] The TV App MUST provide facilities to install and use TV Channels and meet the following requirements:

The TV app that is required for Android device implementations declaring the `android.software.live_tv` feature flag, MUST meet the following requirements:

- Device implementations SHOULD allow third-party TIF-based inputs (third-party inputs) to be installed and managed.
- Device implementations MAY provide visual separation between pre-installed TIF-based inputs (installed inputs) and third-party inputs.
- Device implementations SHOULD NOT display the third-party inputs more than a single navigation action away from the TV App (i.e. expanding a list of third-party inputs from the TV App).

The Android Open Source Project provides an implementation of the TV App that meets the above requirements.

3.12.1.1. Electronic Program Guide

If device implementations support TIF, they:

- [C-1-1] MUST show an informational and interactive overlay, which MUST include an electronic program guide (EPG) generated from the values in the TvContract.Programs fields.
- [C-1-2] On channel change, device implementations MUST display EPG data for the currently playing program.
- [SR] The EPG is STRONGLY RECOMMENDED to display installed inputs and third-party inputs with equal prominence. The EPG SHOULD NOT display the third-party inputs more than a single navigation action away from the installed inputs on the EPG.
- The EPG SHOULD display information from all installed inputs and third-party inputs.
- The EPG MAY provide visual separation between the installed inputs and third-party inputs.

3.12.1.2. Navigation

If device implementations support TIF, they:

- [C-1-1] MUST allow navigation for the following functions via the D-pad, Back, and Home keys on the Android Television device’s input device(s) (i.e. remote control, remote control application, or game controller):
  - Changing TV channels
  - Opening EPG
  - Configuring and tuning to third-party TIF-based inputs
  - Opening Settings menu
- SHOULD pass key events to HDMI inputs through CEC.

3.12.1.3. TV input app linking

If device implementations support TIF, they:

- [C-1-1] Android Television device implementations MUST support TV input app linking , which allows all inputs to provide activity links from the current activity to another activity (i.e. a link from live programming to related content).
- [C-1-2] The TV App MUST show TV input app linking when it is provided.

3.12.1.4. Time shifting

If device implementations support TIF, they:

- [SR] STRONGLY RECOMMENDED to support time shifting, which allows the user to pause and resume live content.
- SHOULD provide the user a way to pause and resume the currently playing program, if time shifting for that program is available .

3.12.1.5. TV recording

If device implementations support TIF, they:

- [SR] STRONGLY RECOMMENDED to support TV recording.
- If the TV input supports recording and the recording of a program is not prohibited , the EPG MAY provide a way to record a program .
- SHOULD provide a user interface to play recorded programs.

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 the UI framework that supports third-party apps that depend on `MediaBrowser` and `MediaSession` , they:

- [C-1-1] MUST display MediaItem icons and notification icons unaltered.
- [C-1-2] MUST display those items as described by MediaSession, e.g., metadata, icons, imagery.
- [C-1-3] MUST show app title.
- [C-1-4] MUST have drawer to present MediaBrowser hierarchy.

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 `"ephemeral"` .
- [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] IInstalled Apps MUST NOT see details about Instant Apps on the device unless the Instant App explicitly connects to the installed application.

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.

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

systemDevice implementations.

- [C-0-2] MUST support verifying ".apk" files using the 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 prompt, 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.

Device implementations 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.

Device implementations MUST have an activity that handles the `android.settings.MANAGE_UNKNOWN_APP_SOURCES` intent. They 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.

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 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 the following audio encoding:

- [C-1-1] PCM/WAVE

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:

- [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-1-5] FLAC
- [C-1-6] MP3
- [C-1-7] MIDI
- [C-1-8] Vorbis
- [C-1-9] PCM/WAVE
- [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

5.1.3. Audio Codecs Details

Format/Codec	Details	Supported File Types/Container Formats
		• 3GPP (.3gp)

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)
MPEG-4 HE AAC Profile (AAC+)	Support for mono/stereo/5.0/5.1 content with standard sampling rates from 16 to 48 kHz.	
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.	
AAC ELD (enhanced low delay AAC)	Support for mono/stereo content with standard sampling rates from 16 to 48 kHz.	
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	
FLAC	Mono/Stereo (no multichannel). Sample rates up to 48 kHz (but up to 44.1 kHz is RECOMMENDED on devices with 44.1 kHz output, as the 48 to 44.1 kHz downsampler does not include a low-pass filter). 16-bit RECOMMENDED; no dither applied for 24-bit.	FLAC (.flac) only
MP3	Mono/Stereo 8-320Kbps constant (CBR) or variable bitrate (VBR)	MP3 (.mp3)
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) • Matroska (.mkv, Android 4.0+)
PCM/WAVE	16-bit linear PCM (rates up to limit of hardware). Devices MUST support sampling rates for raw PCM recording at 8000, 11025, 16000, and 44100 Hz frequencies.	WAVE (.wav)
Opus		Matroska (.mkv), Ogg(.ogg)

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

5.1.5. Image Decoding

See more details in 5.1.6. Image Codecs Details .

Device impelementations MUST support encoding the following image decoding:

- [C-0-1] JPEG
- [C-0-2] GIF
- [C-0-3] PNG
- [C-0-4] BMP
- [C-0-5] WebP
- [C-0-6] Raw

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)

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 flexible color format (COLOR_FormatYUV420Flexible).

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.

5.1.8. Video Codecs List

Format/Codec	Details	Supported File Types/ Container Formats
H.263		• 3GPP (.3gp) • MPEG-4 (.mp4)
H.264 AVC	See section 5.2 and 5.3 for details	• 3GPP (.3gp) • MPEG-4 (.mp4) • MPEG-2 TS (.ts, AAC audio only, not seekable, Android 3.0+)
H.265 HEVC	See section 5.3 for details	MPEG-4 (.mp4)
MPEG-2	Main Profile	MPEG2-TS
MPEG-4 SP		3GPP (.3gp)
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)

5.2. 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.

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.
- SHOULD support writing Matroska WebM files.
- SHOULD use 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:

* SHOULD support writing Matroska WebM files.

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.

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

* [C-2-1] MUST provide a Dolby Vision-capable extractor.
* [C-2-2] MUST properly display Dolby Vision content on the device screen or on a standard video output port (e.g., HDMI).
* [C-2-3] 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.

5.3.1. 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 encoding profiles in the following table.
* [C-2-2] MUST support the HD 1080p video 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	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

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-2-2] 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

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

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 Hz
- **Channels** : Mono
- [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

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 impelementations declare `android.hardware.microphone` and noise suppression (reduction) technologies tuned for speech recognition, they:

- [C-2-1] MUST allow this audio affect to be controllable with the `android.media.audiofx.NoiseSuppressor` API.
- [C-2-2] MUST uniquely identfiy 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`

5.5. 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:
  - **Format** : Linear PCM, 16-bit
  - **Sampling rates** : 8000, 11025, 16000, 22050, 32000, 44100
  - **Channels** : Mono, 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` , `LoudnessEnhancer` .
- [C-1-2] MUST support the visualizer API implementation, controllable through the `Visualizer` class.
- 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` .

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 .

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

- [SR] Cold output latency of 100 milliseconds or less
- [SR] Continuous output latency of 45 milliseconds or less
- [SR] Minimize the cold output jitter

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

- [SR] STRONGLY RECOMMENDED to report low latency audio by declaring `android.hardware.audio.low_latency` feature flag.
- [SR] STRONGLY RECOMMENDED to also meet the requirements for low-latency audio via the AAudio API.

If device implementations do not meet the requirements for low-latency audio via the OpenSL ES

PCM buffer queue API, they:

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

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

- [SR] Cold input latency of 100 milliseconds or less
- [SR] Continuous input latency of 30 milliseconds or less
- [SR] Continuous round-trip latency of 50 milliseconds or less
- [SR] Minimize the cold input jitter

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 Segmant 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 wired external displays.

5.9. Musical Instrument Digital Interface (MIDI)

If a device implementation supports the inter-app MIDI software transport (virtual MIDI devices), and it supports MIDI over *all* of the following MIDI-capable hardware transports for which it provides generic non-MIDI connectivity, it is:

- [SR] STRONGLY RECOMMENDED to report support for feature android.software.midi via the android.content.pm.PackageManager class.

The MIDI-capable hardware transports are:

- USB host mode (section 7.7 USB)
- USB peripheral mode (section 7.7 USB)
- MIDI over Bluetooth LE acting in central role (section 7.4.3 Bluetooth)

If the device implementation provides generic non-MIDI connectivity over a particular MIDI-capable hardware transport listed above, but does not support MIDI over that hardware transport, it:

- [C-1-1] MUST NOT report support for feature android.software.midi.

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 the [OpenSL ES](#) PCM buffer queue API.
- SHOULD provide a sustainable level of CPU performance while audio is active.
- 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 underruns (output) or overruns (input) 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).

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 meet the requirements via the OpenSL ES PCM buffer queue API, they:

- [SR] STRONGLY RECOMMENDED to also meet the same requirements via the [AAudio](#) API.

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, they:

- [C-3-1] MUST have a continuous round-trip audio latency of 20 milliseconds or less.
- The continuous round-trip audio latency SHOULD be 10 milliseconds or less over the USB host mode port using USB audio class.

If device implementations include a USB port(s) supporting USB host mode, they:

- [C-4-1] MUST implement the USB audio class.

If device implementations include an HDMI port, they:

- [C-5-1] MUST support output in stereo and eight channels at 20-bit or 24-bit depth and 192 kHz without bit-depth loss or resampling.

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 all adb functions as documented in the Android SDK including dumpsys .
    - [C-0-3] MUST NOT alter the format or the contents of device system events (batterystats , diskstats, fingerprint, graphicsstats, netstats, notification, procstats) logged via dumpsys.
    - [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. For example:
        - Device implementations without a USB port supporting peripheral mode MUST implement adb via local-area network (such as Ethernet or Wi-Fi).
        - MUST provide drivers for Windows 7, 9 and 10, allowing developers to connect to the device using the adb protocol.
- **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 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.

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 and MUST provide a mechanism to enable Developer Options without the need for any special whitelisting.
- 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 nonphone 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 . Devices 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

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` .

- [C-0-1] Device implementations 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] Device implementations 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.

7.1.1.2. Screen Aspect Ratio

While there is no restriction to the screen aspect ratio value of the physical screen display, the screen aspect ratio of the logical display that third-party apps are rendered within, as can be derived from the height and width values reported through the `view.Display` APIs and Configuration API, MUST meet the following requirements:

- [C-0-1] Device implementations with the `Configuration.uiMode` set as `UI_MODE_TYPE_NORMAL` MUST have an aspect ratio value between 1.3333 (4:3) and 1.86 (roughly 16:9), unless the app can be deemed as ready to be stretched longer by meeting 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 is targeting API level 26 or higher and does not declare a `android:MaxAspectRatio` that would restrict the allowed aspect ratio.
- [C-0-2] 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)
  - 160 dpi (mdpi)
  - 213 dpi (tvdpi)
  - 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 a screen or video output, they:

- [C-1-1] MUST report correct values for all 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 reasonable values for all display metrics 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 1.0 and 2.0, as embodied and detailed in the Android SDK documentation .
- [SR] are STRONGLY RECOMMENDED to support OpenGL ES 3.0.
- SHOULD support OpenGL ES 3.1 or 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` and `EGL_ANDROID_recordable` extensions.
- [SR] are STRONGLY RECOMMENDED to support EGL_KHR_partial_update.
- 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.

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 3.0 or 3.1, they:

- [SR] Are STRONGLY RECOMMENDED to include support for Vulkan 1.0 .

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

- SHOULD include support for Vulkan 1.0.

Device implementations, if including support for Vulkan 1.0:

- [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 enumarate, 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.

Device implementations, if not including support for Vulkan 1.0:

- [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 enumarate any `VkPhysicalDevice` for the Vulkan native API `vkEnumeratePhysicalDevices()` .

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.

- [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 `Display.isWideColorGamut()` , 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 NTSC 1953 in CIE 1931 xyY space.
- [C-1-4] MUST support OpenGL ES 3.0, 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_colorspace_scrgb_linear` , and `EGL_GL_colorspace_display_p3` extensions.
- [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 the display. Devices MUST support all of these APIs as defined by the Android SDK unless specifically allowed in this document.

Device implementations:

- [C-0-1] MUST support displays capable of rendering 16-bit color graphics.
- SHOULD support displays capable of 24-bit color graphics.
- [C-0-2] MUST support displays capable of rendering animations.
- [C-0-3] MUST use the display technology that 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 display 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 a touchscreen 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 preloaded 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:

- [C-0-1] MUST provide the Home function.
- 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.

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), 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, 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 do not include a touchscreen (and rely on a pointer device only) 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` and MUST report only `android.hardware.faketouch` .

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.
* [C-1-7] MUST report `TOUCHSCREEN_NOTOUCH` for the `Configuration.touchscreen` API field.

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

If device implementations declare the `android.hardware.gamepad` feature flag, they:

- [C-1-1] MUST have 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.
- [C-1-2] MUST be capable to map HID events to it's associated Android `view.InputEvent` constants as listed in the below tables. The upstream Android implementation includes implementation for game controllers that satisfies this requirement.

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 streamed with a minimum required latency of 5 ms + 2 * sample_time 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.
- [SR] SHOULD 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. Existing and new Android devices are **STRONGLY RECOMMENDED** to meet these requirements so they will be able to upgrade to the future platform releases where this might become a REQUIRED component. The synchronization error SHOULD be below 100 milliseconds.
- [C-1-7] 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-1-8] 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.

- 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.

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 .

7.3.1. Accelerometer

- Device implementations SHOULD 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 the `TYPE_ACCELEROMETER_UNCALIBRATED` sensor if online accelerometer calibration is available.
- 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.
- SHOULD also implement `TYPE_ACCELEROMETER_UNCALIBRATED` sensor.

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 gyroscope sensor, they:

* [C-3-1] MUST implement the `TYPE_GRAVITY` and `TYPE_LINEAR_ACCELERATION` composite sensors.
* SHOULD implement the `TYPE_GAME_ROTATION_VECTOR` composite sensor.
* [SR] Existing and new Android devices are STRONGLY RECOMMENDED to implement the `TYPE_GAME_ROTATION_VECTOR` sensor.

If device implementations include a 3-axis accelerometer, a gyroscope sensor and a magnetometer sensor, they:

* [C-4-1] MUST implement a `TYPE_ROTATION_VECTOR` composite sensor.

7.3.2. Magnetometer

* Device implementations SHOULD include a 3-axis magnetometer (compass).

If device impelementations 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.
* SHOULD implement `TYPE_MAGNETIC_FIELD_UNCALIBRATED` sensor.
* [SR] Existing and new Android devices are STRONGLY RECOMMENDED to implement the `TYPE_MAGNETIC_FIELD_UNCALIBRATED` sensor.

If device impelementations include a 3-axis magnetometer, an accelerometer sensor and a gyroscope sensor, they:

* [C-2-1] MUST implement a `TYPE_ROTATION_VECTOR` composite sensor.

If device impelementations include a 3-axis magnetometer, an accelerometer, they:

* MAY implement the `TYPE_GEOMAGNETIC_ROTATION_VECTOR` sensor.

If device impelementations 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:

- SHOULD 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).
    - [SR] After making such a location calculation, it is STRONGLY RECOMMENDED for the device to be able to determine its location, in open sky, within 10 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-1-5] MUST report the GNSS technology generation through the test API ‘getGnssYearOfHardware’.
    - [SR] Continue to deliver normal GPS/GNSS location outputs during an emergency phone call.
    - [SR] Report GNSS measurements from all constellations tracked (as reported in GnssStatus messages), with the exception of SBAS.
    - [SR] Report AGC, and Frequency of GNSS measurement.
    - [SR] Report all accuracy estimates (including Bearing, Speed, and Vertical) as part of each GPS Location.
    - [SR] are STRONGLY RECOMMENDED to meet as many as possible from the additional mandatory requirements for devices reporting the year "2016" or "2017" through the Test API `LocationManager.getGnssYearOfHardware()` .

If device implementations include a GPS/GNSS receiver and report the capability to applications through the `android.hardware.location.gps` feature flag and the `LocationManager.getGnssYearOfHardware()` Test API reports the year "2016" or newer, they:

- [C-2-1] MUST report GPS measurements, as soon as they are found, even if a location calculated from GPS/GNSS is not yet reported.
- [C-2-2] MUST report GPS 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 device implementations include a GPS/GNSS receiver and report the capability to applications through the `android.hardware.location.gps` feature flag and the `LocationManager.getGnssYearOfHardware()` Test API reports the year "2017" or newer, they:

- [C-3-1] MUST continue to deliver normal GPS/GNSS location outputs during an emergency phone call.
- [C-3-2] MUST report GNSS measurements from all constellations tracked (as reported in GnssStatus messages), with the exception of SBAS.
- [C-3-3] MUST report AGC, and Frequency of GNSS measurement.
- [C-3-4] MUST report all accuracy estimates (including Bearing, Speed, and Vertical) as part of each GPS Location.

7.3.4. Gyroscope

Device implementations:

- SHOULD include a gyroscope (angular change sensor).
- SHOULD NOT include a gyroscope sensor unless a 3-axis accelerometer is also included.

If device implementations include a 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 SHOULD also implement `TYPE_GYROSCOPE_UNCALIBRATED` sensor.
- [C-1-3] MUST be capable of measuring orientation changes up to 1,000 degrees per second.
- [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] Existing and new Android devices are STRONGLY RECOMMENDED to implement the `SENSOR_TYPE_GYROSCOPE_UNCALIBRATED` sensor.
- [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 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 gyroscope and a accelerometer sensor, they:

- [C-3-1] MUST implement the `TYPE_GRAVITY` and `TYPE_LINEAR_ACCELERATION` composite sensors.
- [SR] Existing and new Android devices are STRONGLY RECOMMENDED to implement the `TYPE_GAME_ROTATION_VECTOR` sensor.
- SHOULD implement the `TYPE_GAME_ROTATION_VECTOR` composite sensor.

7.3.5. Barometer

- Device implementations SHOULD 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

Device implementations: *MAY include an ambient thermometer (temperature sensor).* MAY but SHOULD NOT include a CPU temperature sensor.

If device implementations include an ambient thermometer (temperature sensor), they:

- [C-1-1] MUST be defined as `SENSOR_TYPE_AMBIENT_TEMPERATURE` and MUST measure the ambient (room/vehicle cabin) temperature from where the user is interacting with the device in degrees Celsius.
- [C-1-2] MUST be defined as `SENSOR_TYPE_TEMPERATURE` .
- [C-1-3] MUST measure the temperature of the device CPU.
- [C-1-4] MUST NOT measure any other temperature.

Note the `SENSOR_TYPE_TEMPERATURE` sensor type was deprecated in Android 4.0.

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.
    - MUST have a measurement resolution of at least 1024 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.
    - MUST have a measurement noise not above 400 uG/√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.
    - SHOULD have a stationary noise bias stability of \<15 μg √Hz from 24hr static dataset.
    - SHOULD have a bias change vs. temperature of ≤ +/- 1mg / °C.
    - SHOULD have a best-fit line non-linearity of ≤ 0.5%, and sensitivity change vs. temperature of ≤ 0.03%/C°.
    - SHOULD have white noise spectrum to ensure adequate qualification of sensor's noise integrity.
- [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.
    - MUST have a measurement noise not above 0.014°/s/√Hz.
    - SHOULD have a stationary bias stability of < 0.0002 °/s √Hz from 24-hour static dataset.
    - 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 white noise spectrum to ensure adequate qualification of sensor's noise integrity.
    - SHOULD have calibration error less than 0.002 rad/s in temperature range 10 ~ 40 ℃ when device is stationary.
- [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 uT.

    - 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.
    - SHOULD have white noise spectrum to ensure adequate qualification of sensor's noise integrity.
- [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 which:
    - 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 4 mW.
- [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 sensor and Magnetometer MUST be within 2.5 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. Fingerprint Sensor

If device implementations include a secure lock screen, they:

- SHOULD include a fingerprint sensor.

If device implementations include a fingerprint sensor and make the sensor available to third-party apps, they:

- [C-1-1] MUST declare support for the `android.hardware.fingerprint` feature.
- [C-1-2] MUST fully implement the corresponding API as described in the Android SDK documentation.
- [C-1-3] MUST have a false acceptance rate not higher than 0.002%.
- [C-1-4] MUST rate limit attempts for at least 30 seconds after five false trials for fingerprint verification.
- [C-1-5] MUST have a hardware-backed keystore implementation, and perform the fingerprint matching in a Trusted Execution Environment (TEE) or on a chip with a secure channel to the TEE.
- [C-1-6] MUST have all identifiable fingerprint data encrypted and cryptographically authenticated such that they cannot be acquired, read or altered outside of the Trusted Execution Environment (TEE) as documented in the implementation guidelines on the Android Open Source Project site.
- [C-1-7] MUST prevent adding a fingerprint 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-1-8] MUST NOT enable 3rd-party applications to distinguish between individual

fingerprints.

- [C-1-9] MUST honor the DevicePolicyManager.KEYGUARD_DISABLE_FINGERPRINT flag.
- [C-1-10] MUST, when upgraded from a version earlier than Android 6.0, have the fingerprint data securely migrated to meet the above requirements or removed.
- [SR] STRONGLY RECOMMENDED to have a false rejection rate of less than 10%, as measured on the device.
- [SR] STRONGLY RECOMMENDED to have a latency below 1 second, measured from when the fingerprint sensor is touched until the screen is unlocked, for one enrolled finger.
- SHOULD use the Android Fingerprint icon provided in the Android Open Source Project.

7.3.11. Android Automotive-only sensors

Automotive-specific sensors are defined in the `android.car.CarSensorManager API`.

7.3.11.1. Current Gear

See Section 2.5.1 for device-specific requirements.

7.3.11.2. Day Night Mode

See Section 2.5.1 for device-specific requirements.

7.3.11.3. Driving Status

See Section 2.5.1 for device-specific requirements.

7.3.11.4. Wheel Speed

See Section 2.5.1 for device-specific requirements.

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.

7.4. 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.

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.

7.4.2. 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.
- SHOULD 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
- SHOULD only allow the following information elements in probe request frames, while STA is disconnected:
    - SSID Parameter Set (0)
    - DS Parameter Set (3)

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.
- SHOULD 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 then 30 minutes and whenever Wi-Fi Aware is enabled.

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` .

7.4.3. Bluetooth

If device implementations support Bluetooth Audio profile, they:

- SHOULD support Advanced Audio Codecs and Bluetooth Audio Codecs (e.g. LDAC).

If device implementations 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, AVCP, OBEX, 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.
- 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.
- [SR] STRONGLY RECOMMENDED to implement a Resolvable Private Address (RPA) timeout no longer than 15 minutes and rotate the address at timeout to protect user privacy.

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-1-3] MUST be capable of transmitting and receiving data via the following peer-to-peer standards and protocols:
- ISO 18092
- LLCP 1.2 (defined by the NFC Forum)

- SDP 1.0 (defined by the NFC Forum)
- NDEF Push Protocol
- SNEP 1.0 (defined by the NFC Forum)
- [C-1-4] MUST include support for Android Beam and SHOULD enable Android Beam by default.
- [C-1-5] MUST be able to send and receive using Android Beam, when Android Beam is enabled or another proprietary NFC P2p mode is turned on.
- [C-1-6] MUST implement the SNEP default server. Valid NDEF messages received by the default SNEP server MUST be dispatched to applications using the `android.nfc.ACTION_NDEF_DISCOVERED` intent. Disabling Android Beam in settings MUST NOT disable dispatch of incoming NDEF message.
- [C-1-7] MUST honor the `android.settings.NFCSHARING_SETTINGS` intent to show NFC sharing settings .
- [C-1-8] MUST implement the NPP server. Messages received by the NPP server MUST be processed the same way as the SNEP default server.
- [C-1-9] MUST implement a SNEP client and attempt to send outbound P2P NDEF to the default SNEP server when Android Beam is enabled. If no default SNEP server is found then the client MUST attempt to send to an NPP server.
- [C-1-10] MUST allow foreground activities to set the outbound P2P NDEF message using `android.nfc.NfcAdapter.setNdefPushMessage` , and `android.nfc.NfcAdapter.setNdefPushMessageCallback` , and `android.nfc.NfcAdapter.enableForegroundNdefPush` .
- SHOULD use a gesture or on-screen confirmation, such as 'Touch to Beam', before sending outbound P2P NDEF messages.
- [C-1-11] MUST support NFC Connection handover to Bluetooth when the device supports Bluetooth Object Push Profile.
- [C-1-12] MUST support connection handover to Bluetooth when using `android.nfc.NfcAdapter.setBeamPushUris` , by implementing the " Connection Handover version 1.2 " and " Bluetooth Secure Simple Pairing Using NFC version 1.0 " specs from the NFC Forum. Such an implementation MUST implement the handover LLCP service with service name "urn:nfc:sn:handover" for exchanging the handover request/select records over NFC, and it MUST use the Bluetooth Object Push Profile for the actual Bluetooth data transfer. For legacy reasons (to remain compatible with Android 4.1 devices), the implementation SHOULD still accept SNEP GET requests for exchanging the handover request/select records over NFC. However an implementation itself SHOULD NOT send SNEP GET requests for performing connection handover.
- [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] (https://developer.android.com/reference/android/nfc/cardemulation/NfcFCardEmulation.html) 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. 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 200Kbit/sec or greater. Examples of technologies that satisfy this requirement include EDGE, HSPA, EV-DO, 802.11g, Ethernet, Bluetooth PAN, etc.
- [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.
- 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.

The required level of IPv6 support depends on the network type, as follows:
If devices implementations support Wi-Fi networks, they:

- [C-1-1] MUST support dual-stack and IPv6-only operation on Wi-Fi.

If device impelementations support Ethernet networks, they:

- [C-2-1] MUST support dual-stack operation on Ethernet.

If device implementations support cellular data, they:

- [C-3-1] MUST simultaneously meet these requirements on each network to which it is connected when a device is simultaneously connected to more than one network (e.g., Wi-Fi and cellular data), .

- SHOULD support IPv6 operation (IPv6-only and possibly dual-stack) on cellular data.

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
- [C-1-2] 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 whitelist.

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` .
- [C-2-3] MUST have an activity that handles the `Settings.ACTION_IGNORE_BACKGROUND_DATA_RESTRICTIONS_SETTINGS` intent but MAY implement it as a no-op.

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.

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-1-5] 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 the current application does not explicitly request that the Camera display be rotated via a call to the android.hardware.Camera.setDisplayOrientation() method.
- [C-1-6] MUST NOT mirror the final captured still image or video streams returned to application callbacks or committed to media storage.
- [C-1-7] 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 impelmentations 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 port.
- 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 supported, 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.

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` .
- [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 on the `android.hardware.Camera.Parameters` 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.

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 enforce the `android.permission.WRITE_EXTERNAL_STORAGE` permission on this shared storage as documented in the SDK. Shared storage MUST otherwise be writable by any application that obtains that permission.

Device implementations MAY meet the above requirements using either:

- a 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 protion 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 include multiple shared storage paths (such as both an SD card slot and shared internal storage), they:

- [C-3-1] MUST allow only pre-installed and privileged Android applications with the `WRITE_EXTERNAL_STORAGE` permission to write to the secondary external storage, except when writing to their package-specific directories or within the `URI` returned by firing the `ACTION_OPEN_DOCUMENT_TREE` intent.

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 including a USB port, 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).
* [SR] 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] 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 a device implementation includes one or more analog audio ports, at least one of the audio port(s) SHOULD 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.
- [SR] STRONGLY RECOMMENDED to 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`
- SHOULD support audio plugs with the OMTP pin-out order.
- SHOULD 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.

7.8.3. 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.

7.9. 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 High Performance

If device implementations identify the support of high performance VR for longer user periods through the `android.hardware.vr.high_performance` feature flag, they:

- [C-1-1] MUST have at least 2 physical cores.
- [C-1-2] MUST declare `android.software.vr.mode feature`.
- [C-1-3] MUST support sustained performance mode.
- [C-1-4] MUST support OpenGL ES 3.2.
- [C-1-5] MUST support Vulkan Hardware Level 0 and SHOULD support Vulkan Hardware Level 1.
- [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` , and expose the extensions in the list of available EGL extensions.
- [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-1-8] MUST implement `GL_EXT_multisampled_render_to_texture` , `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-1-9] MUST implement support for `AHardwareBuffer` flags `AHARDWAREBUFFER_USAGE_GPU_DATA_BUFFER` and `AHARDWAREBUFFER_USAGE_SENSOR_DIRECT_DATA` as described in the NDK.
- [C-1-10] MUST implement support for `AHardwareBuffers` with more than one layer.
- [C-1-11] MUST support H.264 decoding at least 3840x2160@30fps-40Mbps (equivalent to 4 instances of 1920x1080@30fps-10Mbps or 2 instances of 1920x1080@60fps-20Mbps).
- [C-1-12] MUST support HEVC and VP9, MUST be capable to decode at least 1920x1080@30fps-10Mbps and SHOULD be capable to decode 3840x2160@30fps-20Mbps (equivalent to 4 instances of 1920x1080@30fps-5Mbps).
- [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 be FullHD(1080p) and STRONGLY RECOMMENDED TO BE be QuadHD (1440p) or higher.
- [C-1-15] The display MUST measure between 4.7" and 6.3" diagonal.
- [C-1-16] The display MUST update at least 60 Hz while in VR Mode.
- [C-1-17] The display latency on Gray-to-Gray, White-to-Black, and Black-to-White switching time MUST be ≤ 3 ms.
- [C-1-18] The display MUST support a low-persistence mode with ≤5 ms persistence,

persistence being defined as the amount of time for which a pixel is emitting light.
- [C-1-19] MUST support Bluetooth 4.2 and Bluetooth LE Data Length Extension section 7.4.3 .
- [SR] STRONGLY RECOMMENDED to support `android.hardware.sensor.hifi_sensors` feature and MUST meet the gyroscope, accelerometer, and magnetometer related requirements for `android.hardware.hifi_sensors` .
- 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 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.

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

Android includes App Standby and Doze power-saving modes to optimize battery usage. *[SR] All Apps exempted from these modes are STRONGLY RECOMMENDED to be made visible to the end user.* [SR] The triggering, maintenance, wakeup algorithms and the use of global system settings of these power-saving modes are STRONGLY RECOMMENDED NOT to deviate from the Android Open Source Project.

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 implements S3 and S4 power states as defined by the ACPI, they:

- [C-1-1] MUST only enter these states when closing a lid that is physically part of the device.

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 preloaded 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

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

- [C-1-1] 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-2-1] 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.

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. Kernel Security Features

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 protections (e.g. `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`).
- [SR] STRONGLY RECOMMENDED to keep kernel data which is written only during initialization marked read-only after initialization (e.g. `__ro_after_init`).
- [SR} STRONGLY RECOMMENDED to implement static and dynamic object size bounds checking of copies between user-space and kernel-space (e.g. `CONFIG_HARDENED_USERCOPY`).
- [SR] STRONGLY RECOMMENDED to never execute user-space memory when running in the kernel (e.g. hardware PXN, or emulated via `CONFIG_CPU_SW_DOMAIN_PAN` or `CONFIG_ARM64_SW_TTBR0_PAN`).
- [SR] STRONGLY RECOMMENDED to never 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`).

- [SR] 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`]).

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.
- 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 an mandatory access control system that is equivalent to SELinux.

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-1-1] 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.

9.8.2. Recording

If device implementations include functionality in the system that captures the contents displayed on the screen and/or records the audio stream played on the device, 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 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.

9.9. Data Storage Encryption

If device implementations support a secure lock screen as described in section 9.11.1 , they:

* [C-1-1] MUST support data storage encryption of 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.

If device implementations support a secure lock screen as described in section 9.11.1 and support data storage encryption with Advanced Encryption Standard (AES) crypto performance above 50MiB/sec, they:

* [C-2-1] MUST enable the data storage encryption by default at the time the user has completed the out-of-box setup experience. If device implementations are already launched on an earlier Android version with encryption disabled by default, such a device cannot meet the requirement through a system software update and thus MAY be exempted.
* SHOULD meet the above data storage encryption requirement via implementing File Based Encryption (FBE).

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. File Based Encryption

If device implementations support FBE, 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-1-3] MUST NOT offer any method to unlock the CE protected storage without the user-supplied credentials.
* [C-1-4] MUST support Verified Boot and ensure that DE keys are cryptographically bound to the device's hardware root of trust.
* [C-1-5] MUST support encrypting file contents using AES with a key length of 256-bits in XTS mode.
* [C-1-6] MUST support encrypting file name using AES with a key length of 256-bits in CBC-CTS mode.
* The keys protecting CE and DE storage areas:
* [C-1-7] MUST be cryptographically bound to a hardware-backed Keystore.
* [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.
* SHOULD make preloaded essential apps (e.g. Alarm, Phone, Messenger) Direct Boot aware.
* MAY support alternative ciphers, key lengths and modes for file content and file name encryption, but MUST use the mandatorily supported ciphers, key lengths and modes by default.

The upstream Android Open Source project provides a preferred implementation of this feature based on the Linux kernel ext4 encryption feature.

9.9.3. Full Disk Encryption

If device implementations support full disk encryption (FDE), they:

* [C-1-1] MUST use AES with a key of 128-bits (or greater) and a mode designed for storage (for example, AES-XTS, AES-CBC-ESSIV).
* [C-1-2] MUST use a default passcode to wrap the encryption key and MUST NOT write the encryption key to storage at any time without being encrypted.
* [C-1-3] MUST provide the user the possibility to AES encrypt the encryption key, except when it is in active use, with the lock screen credentials stretched using a slow stretching algorithm (e.g. PBKDF2 or scrypt).

- [C-1-4] The above default password stretching algorithm MUST be cryptographically bound to that keystore when the user has not specified a lock screen credentials or has disabled use of the passcode for encryption and the device provides a hardware-backed keystore.
- [C-1-5] MUST NOT send encryption key off the the device (even when wrapped with the user passcode and/or hardware bound key).

The upstream Android Open Source project provides a preferred implementation of this feature, based on the Linux kernel feature dm-crypt.

9.10. Device Integrity

The following requirements ensures there is transparancy 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.

Verified boot is a feature that guarantees the integrity of the device software. If a device implementation supports the feature, it:

- [C-1-1] MUST declare the platform feature flag `android.software.verified_boot`.
- [C-2-1] MUST perform verification on every boot sequence.
- [C-3-1] 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-4-1] 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-5-1] MUST use verification algorithms as strong as current recommendations from NIST for hashing algorithms (SHA-256) and public key sizes (RSA-2048).
- [C-6-1] 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-7-1] MUST NOT allow verified partitions on the device to be modified unless the user has explicitly unlocked the boot loader.
- [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.
- [SR] STRONGLY RECOMMENDED to use tamper-evident storage: for when the bootloader is unlocked. Tamper-evident storage means that the boot loader can detect if the storage has been tampered with from inside the HLOS (High Level Operating System).
- [SR] STRONGLY RECOMMENDED to prompt the user, while using the device, and require physical confirmation before allowing a transition from boot loader locked mode to boot loader unlocked mode.
- [SR] STRONGLY RECOMMENDED to implement rollback protection for the HLOS (e.g. boot, system partitions) and to use tamper-evident storage for storing the metadata used for determining the minimum allowable OS version.
- 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.

The upstream Android Open Source Project provides a preferred implementation of this feature in the `external/avb/` repository, which can be integrated into the boot loader used for loading Android.

Device implementations with Advanced Encryption Standard (AES) crypto performance above 50 MiB/seconds:

- [C-8-1] MUST support verified boot for device integrity.

If a device implementation is already launched without supporting verified boot on an earlier version of Android, such a device can not add support for this feature with a system software update and thus are exempted from the requirement.

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 at least allow more than 8,192 keys to be imported.
- [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 secure hardware.
- [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. 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 hardware-backed keystore, unless it declares the `android.hardware.fingerprint` feature which requires a hardware-backed keystore.

9.11.1. Secure Lock Screen

If device implementations have a secure lock screen and include one or more trust agent, which implements the `TrustAgentService` System API, then they:

- [C-1-1] MUST indicate the user in the Settings and Lock screen user interface of situations where either the screen auto-lock is deferred or the screen lock can be unlocked by the trust agent. The AOSP meets the requirement by showing a text description for the "Automatically lock setting" and "Power button instantly locks setting" menus and a distinguishable icon on the lock screen.
- [C-1-2] MUST respect and fully implement all trust agent APIs in the `DevicePolicyManager` class, such as the `KEYGUARD_DISABLE_TRUST_AGENTS` constant.
- [C-1-3] MUST NOT fully implement the `TrustAgentService.addEscrowToken()` function on a device that is used as the primary personal device (e.g. handheld) but MAY fully implement the function on device implementations typically shared.
- [C-1-4] MUST encrypt the tokens added by `TrustAgentService.addEscrowToken()` before storing them on the device.
- [C-1-5] MUST NOT store the encryption key on the device.
- [C-1-6] MUST inform the user about the security implications before enabling the escrow token to decrypt the data storage.

If device implementations add or modify the authentication methods to unlock the lock screen, then for such an authentication method to be treated as a secure way to lock the screen, they:

- [C-2-1] MUST be the user authentication method as described in Requiring User Authentication For Key Use .
- [C-2-2] MUST unlock all keys for a third-party developer app to use when the user unlocks the secure lock screen. For example, all keys MUST be available for a third-party developer app through relevant APIs, such as `createConfirmDeviceCredentialIntent` and `setUserAuthenticationRequired` .

If device implementations add or modify the authentication methods to unlock the lock screen if based on a known secret then for such an authentication method to be treated as a secure way to lock the screen, they:

- [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] MUST not replace any of the existing authentication methods (PIN,pattern, password) implemented and provided in AOSP.
- [C-3-4] 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` .

If device implementations add or modify the authentication methods to unlock the lock screen if based on a physical token or the location, then for such an authentication method to be treated as a secure way to lock the screen, they:

- [C-4-1] MUST have a fall-back mechanism to use one of the primary authentication methods which is based on a known secret and meets the requirements to be treated as a secure lock screen.
- [C-4-2] MUST be disabled and only allow the primary authentication to unlock the screen when the Device Policy Controller (DPC) application has set the policy with either the `DevicePolicyManager.setKeyguardDisabledFeatures(KEYGUARD_DISABLE_TRUST_AGENT` method or the `DevicePolicyManager.setPasswordQuality()` method with a more

restrictive quality constant than PASSWORD_QUALITY_UNSPECIFIED .

- [C-4-3] The user MUST be challenged for the primary authentication (e.g.PIN, pattern, password) at least once every 72 hours or less.

If device implementations add or modify the authentication methods to unlock the lock screen based on biometrics, then for such an authentication method to be treated as a secure way to lock the screen, they:

- [C-5-1] MUST have a fall-back mechanism to use one of the primary authentication methods which is based on a known secret and meets the requirements to be treated as a secure lock screen.
- [C-5-2] MUST be disabled and only allow the primary authentication to unlock the screen when the Device Policy Controller (DPC) application has set the keguard feature policy by calling the method DevicePolicyManager.setKeyguardDisabledFeatures(KEYGUARD_DISABLE_FINGERPRINT .
- [C-5-3] MUST have a false acceptance rate that is equal or stronger than what is required for a fingerprint sensor as described in section 7.3.10, or otherwise MUST be disabled and only allow the primary authentication to unlock the screen 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_BIOMETRIC_WEAK .
- [C-5-4] The user MUST be challenged for the primary authentication (e.g.PIN, pattern, password) at least once every 72 hours or less.

If device implementations add or modify the authentication methods to unlock the lock screen and if such an authentication method will be used to unlock the keyguard, but will not be treated as a secure lock screen, then they:

- [C-6-1] MUST return false for both the KeyguardManager.isKeyguardSecure() and the KeyguardManager.isDeviceSecure() methods.
- [C-6-2] 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-6-3] MUST NOT reset the password expiration timers set by DevicePolicyManager.setPasswordExpirationTimeout() .
- [C-6-4] MUST NOT authenticate access to keystores if the application has called KeyGenParameterSpec.Builder.setUserAuthenticationRequired(true)).

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 user-generated data. That is, all data except for the following:
  - The system image
  - Any operating system files required by the system image
- [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.

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 MUST pass the Android Compatibility Test Suite (CTS) available from the Android Open Source Project, using the final shipping software on the device. Additionally, device implementers SHOULD use the reference implementation in the Android Open Source tree as much as possible, and 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 8.0. Device implementations MUST pass the latest CTS version available at the time the device software is completed.

10.2. CTS Verifier

Device implementations MUST correctly execute all applicable cases in the 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.

The CTS Verifier has tests for many kinds of hardware, including some hardware that is optional. Device implementations 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.

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

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.

However, if the device implementation includes support for an unmetered data connection such as 802.11 or Bluetooth PAN (Personal Area Network) profile, it MUST support OTA downloads with offline update via reboot.

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.

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, 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. To facilitate this, the system update subsystem for devices that report android.software.device_admin 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.