

android

Compatibility Definition



Android 6.0

Last updated: October 16th, 2015

Copyright © 2015, Google Inc. All rights reserved.

compatibility@android.com

Table of Contents

1. Introduction

2. Device Types

2.1 Device Configurations

3. Software

3.1. Managed API Compatibility

3.2. Soft API Compatibility

3.2.1. Permissions

3.2.2. Build Parameters

3.2.3. Intent Compatibility

3.2.3.1. Core Application Intents

3.2.3.2. Intent Resolution

3.2.3.3. Intent Namespaces

3.2.3.4. Broadcast Intents

3.2.3.5. Default App Settings

3.3. Native API Compatibility

3.3.1. Application Binary Interfaces

3.3.2. 32-bit ARM Native Code Compatibility

3.4. Web Compatibility

3.4.1. WebView Compatibility

3.4.2. Browser Compatibility

3.5. API Behavioral Compatibility

3.6. API Namespaces

3.7. Runtime Compatibility

3.8. User Interface Compatibility

3.8.1. Launcher (Home Screen)

3.8.2. Widgets

3.8.3. Notifications

3.8.4. Search

3.8.5. Toasts

3.8.6. Themes

3.8.7. Live Wallpapers

3.8.8. Activity Switching

3.8.9. Input Management

3.8.10. Lock Screen Media Control

3.8.11. Dreams

3.8.12. Location

3.8.13. Unicode and Font

3.9. Device Administration

3.9.1 Device Provisioning

3.9.1.1 Device Owner provisioning

3.9.1.2 Managed profile provisioning

3.9.2. Managed Profile Support

3.10. Accessibility

3.11. Text-to-Speech

3.12. TV Input Framework

3.12.1. TV App

3.12.1.1. Electronic Program Guide

3.12.1.2. Navigation

3.12.1.3. TV input app linking

4. Application Packaging Compatibility

5. Multimedia Compatibility

5.1. Media Codecs

5.1.1. Audio Codecs

5.1.2. Image Codecs

5.1.3. Video Codecs

5.2. Video Encoding

5.3. Video Decoding

5.4. Audio Recording

5.4.1. Raw Audio Capture

5.4.2. Capture for Voice Recognition

5.4.3. Capture for Rerouting of Playback

5.5. Audio Playback

5.5.1. Raw Audio Playback

5.5.2. Audio Effects

5.5.3. Audio Output Volume

5.6. Audio Latency

5.7. Network Protocols

5.8. Secure Media

5.9. Musical Instrument Digital Interface (MIDI)

5.10. Professional Audio

6. Developer Tools and Options Compatibility

6.1. Developer Tools

6.2. Developer Options

7. Hardware Compatibility

7.1. Display and Graphics

7.1.1. Screen Configuration

7.1.1.1. Screen Size

7.1.1.2. Screen Aspect Ratio

7.1.1.3. Screen Density

7.1.2. Display Metrics

7.1.3. Screen Orientation

7.1.4. 2D and 3D Graphics Acceleration

7.1.5. Legacy Application Compatibility Mode

7.1.6. Screen Technology

7.1.7. Secondary Displays

7.2. Input Devices

7.2.1. Keyboard

7.2.2. Non-touch Navigation

7.2.3. Navigation Keys

7.2.4. Touchscreen Input

7.2.5. Fake Touch Input

7.2.6. Game Controller Support

7.2.6.1. Button Mappings

7.2.7. Remote Control

7.3. Sensors

7.3.1. Accelerometer

7.3.2. Magnetometer

7.3.3. GPS

7.3.4. Gyroscope

7.3.5. Barometer

7.3.6. Thermometer

7.3.7. Photometer

7.3.8. Proximity Sensor

7.3.9. High Fidelity Sensors

7.3.10. Fingerprint Sensor

7.4. Data Connectivity

7.4.1. Telephony

7.4.2. IEEE 802.11 (Wi-Fi)

7.4.2.1. Wi-Fi Direct

7.4.2.2. Wi-Fi Tunneled Direct Link Setup

7.4.3. Bluetooth

7.4.4. Near-Field Communications

7.4.5. Minimum Network Capability

7.4.6. Sync Settings

7.5. Cameras

7.5.1. Rear-Facing Camera

7.5.2. Front-Facing Camera

7.5.3. External Camera

7.5.4. Camera API Behavior

7.5.5. Camera Orientation

7.6. Memory and Storage

7.6.1. Minimum Memory and Storage

7.6.2. Application Shared Storage

7.6.3. Adoptable Storage

7.7. USB

7.8. Audio

7.8.1. Microphone

7.8.2. Audio Output

7.8.2.1. Analog Audio Ports

7.8.3. Near-Ultrasound

8. Performance and Power

8.1. User Experience Consistency

8.2. File I/O Access Performance

8.3. Power-Saving Modes

8.4. Power Consumption Accounting

9. Security Model Compatibility

- 9.1. Permissions
- 9.2. UID and Process Isolation
- 9.3. Filesystem Permissions
- 9.4. Alternate Execution Environments
- 9.5. Multi-User Support
- 9.6. Premium SMS Warning
- 9.7. Kernel Security Features
- 9.8. Privacy
- 9.9. Full-Disk Encryption
- 9.10. Verified Boot
- 9.11. Keys and Credentials
- 9.12. Data Deletion

10. Software Compatibility Testing

- 10.1. Compatibility Test Suite
- 10.2. CTS Verifier

11. Updatable Software

12. Document Changelog

13. Contact Us

14. Resources

1. Introduction

This document enumerates the requirements that must be met in order for devices to be compatible with Android 6.0.

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 [[Resources, 1](#)].

As used in this document, a “device implementer” or “implementer” is a person or organization developing a hardware/software solution running Android 6.0. A “device implementation” or “implementation is the hardware/software solution so developed.

To be considered compatible with Android 6.0, 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 [[Resources, 2](#)] 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 listed in [section 14](#) are derived directly or indirectly from the Android SDK, and will be functionally identical to the information in that SDK’s documentation. For any case 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 references included in [section 14](#) are considered by inclusion to be part of this Compatibility Definition.

2. Device Types

While the Android Open Source Project has been used in the implementation of a variety of device types and form factors, many aspects of the architecture and compatibility requirements were optimized for handheld devices. Starting from Android 5.0, the Android Open Source Project aims to embrace a wider variety of device types as described in this section.

Android Handheld device refers to an Android device implementation that is typically used by holding it in the hand, such as mp3 players, phones, and tablets. Android Handheld device implementations:

- MUST have a touchscreen embedded in the device.
- MUST have a power source that provides mobility, such as a battery.

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 Television devices:

- MUST have an embedded screen OR include a video output port, such as VGA, HDMI, or a wireless port for display.
- MUST declare the features `android.software.leanback` and `android.hardware.type.television` [[Resources, 3](#)].

Android Watch device refers to an Android device implementation intended to be worn on the body,

perhaps on the wrist, and:

- MUST have a screen with the physical diagonal length in the range from 1.1 to 2.5 inches.
- MUST declare the feature `android.hardware.type.watch`.
- MUST support `uiMode = UI_MODE_TYPE_WATCH` [[Resources, 4](#)].

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 Automotive implementations:

- MUST declare the feature `android.hardware.type.automotive`.
- MUST support `uiMode = UI_MODE_TYPE_CAR` [[Resources, 5](#)].

All Android device implementations that do not fit into any of the above device types still MUST meet all requirements in this document to be Android 6.0 compatible, unless the requirement is explicitly described to be only applicable to a specific Android device type from above.

2.1 Device Configurations

This is a summary of major differences in hardware configuration by device type. (Empty cells denote a “MAY”). Not all configurations are covered in this table; see relevant hardware sections for more detail.

Category	Feature	Section	Handheld	Television	Watch	Automotive	Other
Input	D-pad	7.2.2. Non-touch Navigation		MUST			
	Touchscreen	7.2.4. Touchscreen input	MUST		MUST		SHOULD
	Microphone	7.8.1. Microphone	MUST	SHOULD	MUST	MUST	SHOULD
Sensors	Accelerometer	7.3.1. Accelerometer	SHOULD		SHOULD		SHOULD
	GPS	7.3.3. GPS	SHOULD			SHOULD	
Connectivity	Wi-Fi	7.4.2. IEEE 802.11	SHOULD	MUST		SHOULD	SHOULD
	Wi-Fi Direct	7.4.2.1. Wi-Fi Direct	SHOULD	SHOULD			SHOULD
	Bluetooth	7.4.3. Bluetooth	SHOULD	MUST	MUST	MUST	SHOULD
	Bluetooth Low Energy	7.4.3. Bluetooth	SHOULD	MUST	SHOULD	SHOULD	SHOULD
	USB peripheral/host mode	7.7. USB	SHOULD			SHOULD	SHOULD
Output	Speaker and/or Audio output ports	7.8.2. Audio Output	MUST	MUST		MUST	MUST

3. Software

3.1. Managed API Compatibility

The managed Dalvik bytecode execution environment is the primary vehicle for Android applications. The Android application programming interface (API) is the set of Android platform interfaces exposed to applications running in the managed runtime environment. Device implementations MUST provide complete implementations, including all documented behaviors, of any documented API exposed by the Android SDK [[Resources, 6](#)] or any API decorated with the “@SystemApi” marker in the upstream Android source code.

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.

This Compatibility Definition permits some types of hardware for which Android includes APIs to be omitted by device implementations. In such cases, the APIs MUST still be present and behave in a reasonable way. See [section 7](#) for specific requirements for this scenario.

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

Device implementers MUST support and enforce all permission constants as documented by the Permission reference page [[Resources, 7](#)]. 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 [[Resources, 8](#)] that are intended to describe the current device. 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 [Resources, 9].
VERSION.SDK	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 6.0, this field MUST have the integer value 23.
VERSION.SDK_INT	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 6.0, this field MUST have the integer value 23.
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_-]+\$".
FINGERPRINT	<p>A string that uniquely identifies this build. It SHOULD be reasonably human-readable. It MUST follow this template:</p> <pre>\$(BRAND)/\$(PRODUCT)/ \$(DEVICE):\$(VERSION.RELEASE)/\$(ID)/\$(VERSION.INCREMENTAL):\$(TYPE)/\$(TAGS)</pre> <p>For example:</p> <pre>acme/myproduct/ mydevice:6.0/LMYXX/3359:userdebug/test-keys</pre> <p>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.</p>
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 ("").
	An identifier chosen by the device implementer to refer to a specific

ID	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 (“”).
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 (“”).
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._-]+\$”.
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	An value indicating the security patch level of a build. It MUST signify that the build includes all security patches issued up through the designated Android Public Security Bulletin. It MUST be in the format, [YYYY-MM-DD], matching the Public Security Bulletin's broadcast date, for example [2015-10-01].
BASE_OS	An 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 (“”).

3.2.3. Intent Compatibility

Device implementations MUST honor Android’s loose-coupling intent system, as described in the sections below. By “honored ” it is meant that the device implementer MUST provide an Android Activity or Service that specifies a matching intent filter that binds to and implements correct behavior

for each specified intent pattern.

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. The core Android applications are:

- Desk Clock
- Browser
- Calendar
- Contacts
- Gallery
- GlobalSearch
- Launcher
- Music
- Settings

Device implementations SHOULD include the core Android applications as appropriate but MUST include a component implementing the same intent patterns defined by all the “public” Activity or Service components of these core Android applications. Note that Activity or Service components are considered “public” when the attribute `android:exported` is absent or has the value `true`.

3.2.3.2. Intent Resolution

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; device implementers MUST NOT attach special privileges to system applications' use of these intent patterns, or prevent third-party applications from binding to and assuming control of these patterns. This prohibition specifically includes but is not limited to disabling the “Chooser” user interface that allows the user to select between multiple applications that all handle the same intent pattern.

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 (eg. `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 [[Resources, 140](#)]. When such authoritative declarations are defined in an app's intent filter patterns, device implementations:

- MUST attempt to validate any intent filters by performing the validation steps defined in the Digital Asset Links specification [[Resources, 141](#)] as implemented by the Package Manager in the upstream Android Open Source Project.
- 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:
 - 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.
 - 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.
 - 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

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. 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. 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. Android-compatible devices MUST broadcast the public broadcast intents in response to appropriate system events. Broadcast intents are 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.

Device implementations:

- MUST honor the android.settings.HOME_SETTINGS intent to show a default app settings menu for Home Screen, if the device implementation reports android.software.home_screen [[Resources, 10](#)]
- 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 the device implementation reports android.hardware.telephony [[Resources, 11](#)]
- MUST honor the android.settings.NFC_PAYMENT_SETTINGS intent to show a default app settings menu for Tap and Pay, if the device implementation reports android.hardware.nfc.hce [[Resources, 10](#)]

3.3. Native API Compatibility

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 **MUST** be compatible with one or more defined ABIs, and **MUST** implement compatibility with the Android NDK, as below.

If a device implementation includes support for an Android ABI, it:

- **MUST** include support for code running in the managed environment to call into native code, using the standard Java Native Interface (JNI) semantics
- **MUST** be source-compatible (i.e. header compatible) and binary-compatible (for the ABI) with each required library in the list below
- **MUST** support the equivalent 32-bit ABI if any 64-bit ABI is supported
- **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
- **MUST** report, via the above parameters, only those ABIs documented and described in the latest version of the Android NDK ABI Management documentation [[Resources, 12](#)], and **MUST** include support for the Advanced SIMD (a.k.a. NEON) [[Resources, 13](#)] extension
- **SHOULD** be built using the source code and header files available in the upstream Android Open Source Project

The following native code APIs **MUST** be available to apps that include native code:

- libc (C library)
- libm (math library)
- Minimal support for C++
- JNI interface
- liblog (Android logging)
- libz (Zlib compression)
- libdl (dynamic linker)
- libGLESv1_CM.so (OpenGL ES 1.x)
- libGLESv2.so (OpenGL ES 2.0)
- libGLESv3.so (OpenGL ES 3.x)
- libEGL.so (native OpenGL surface management)
- libjnigraphics.so
- libOpenSLES.so (OpenSL ES 1.0.1 audio support)
- libOpenMAXAL.so (OpenMAX AL 1.0.1 support)
- libandroid.so (native Android activity support)
- libmediandk.so (native media APIs support)
- Support for OpenGL, as described below

Note that future releases of the Android NDK may introduce support for additional ABIs. If a device implementation is not compatible with an existing predefined ABI, it **MUST NOT** report support for any ABIs at all.

Note that device implementations **MUST** include `libGLESv3.so` and it **MUST** symlink (symbolic link) to `libGLESv2.so`. In turn, **MUST** export all the OpenGL ES 3.1 and Android Extension Pack [[Resources, 14](#)] function symbols as defined in the NDK release android-21. Although all the symbols must be

present, only the corresponding functions for OpenGL ES versions and extensions actually supported by the device must be fully implemented.

Device implementations MUST NOT include a native library with the name libvulkan.so.

Native code compatibility is challenging. For this reason, device implementers are **STRONGLY RECOMMENDED** to use the implementations of the libraries listed above from the upstream Android Open Source Project.

3.3.2. 32-bit ARM Native Code Compatibility

The ARMv8 architecture deprecates several CPU operations, including some operations used in existing native code. On 64-bit ARM devices, 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

Legacy versions of the Android NDK used `/proc/cpuinfo` to discover CPU features from 32-bit ARM native code. For compatibility with applications built using this NDK, devices MUST include the following lines in `/proc/cpuinfo` when it is read by 32-bit ARM applications:

- "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)

These requirements only apply when `/proc/cpuinfo` is read by 32-bit ARM applications. 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

Android Watch devices MAY, but all other device implementations MUST provide a complete implementation of the `android.webkit.WebView` API.

The platform feature `android.software.webview` MUST be reported on any device that provides a complete implementation of the `android.webkit.WebView` API, and MUST NOT be reported on devices without a complete implementation of the API. The Android Open Source implementation uses code from the Chromium Project to implement the `android.webkit.WebView` [[Resources, 15](#)]. Because it is not feasible to develop a comprehensive test suite for a web rendering system, device implementers MUST use the specific upstream build of Chromium in the `WebView` implementation. Specifically:

- Device `android.webkit.WebView` implementations MUST be based on the Chromium build from the upstream Android Open Source Project for Android 6.0. This build includes a specific set of functionality and security fixes for the `WebView` [[Resources, 16](#)].
- 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 [[Resources, 17](#)].

3.4.2. Browser Compatibility

Android Television, Watch, and Android Automotive implementations MAY omit a browser application, but MUST support the public intent patterns as described in [section 3.2.3.1](#). All other types of device implementations MUST include a standalone Browser application for general user web browsing.

The standalone Browser MAY be based on a browser technology other than WebKit. However, even if an alternate Browser application is used, the android.webkit.WebView component provided to third-party applications MUST be based on WebKit, as described in [section 3.4.1](#).

Implementations MAY ship a custom user agent string in the standalone Browser application.

The standalone Browser application (whether based on the upstream WebKit Browser application or a third-party replacement) SHOULD include support for as much of HTML5 [[Resources, 17](#)] as possible. Minimally, device implementations MUST support each of these APIs associated with HTML5:

- application cache/offline operation [[Resources, 18](#)]
- the <video> tag [[Resources, 19](#)]
- geolocation [[Resources, 20](#)]

Additionally, device implementations MUST support the HTML5/W3C webstorage API [[Resources, 21](#)], and SHOULD support the HTML5/W3C IndexedDB API [[Resources, 22](#)]. 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.

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 [[Resources, 2](#)]. Some specific areas of compatibility are:

- Devices MUST NOT change the behavior or semantics of a standard intent.
- Devices MUST NOT alter the lifecycle or lifecycle semantics of a particular type of system component (such as Service, Activity, ContentProvider, etc.).
- Devices MUST NOT change the semantics of a standard permission.

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.*

Prohibited modifications include:

- Device implementations **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.
- Device implementers **MAY** modify the underlying implementation of the APIs, but such modifications **MUST NOT** impact the stated behavior and Java-language signature of any publicly exposed APIs.
- Device implementers **MUST NOT** add any publicly exposed elements (such as classes or interfaces, or fields or methods to existing classes or interfaces) to the APIs above.

A “publicly exposed element” is any construct which is not decorated with the “@hide” marker as used in the upstream Android source code. In other words, device implementers **MUST NOT** expose new APIs or alter existing APIs in the namespaces noted above. Device implementers **MAY** make internal-only modifications, but those modifications **MUST NOT** be advertised or otherwise exposed to developers.

Device implementers **MAY** add custom APIs, but any such APIs **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. Additionally, if a device implementation includes custom APIs outside the standard Android namespace, those APIs **MUST** be packaged in an Android shared library so that only apps that explicitly use them (via the `lt;uses-librarygt;` 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 **MUST** support the full Dalvik Executable (DEX) format and Dalvik bytecode specification and semantics [[Resources, 23](#)]. Device implementers **SHOULD** use ART, the reference upstream implementation of the Dalvik Executable Format, and the reference implementation's package management system.

Device implementations **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.)

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)
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)
	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). Device implementations that allow third-party applications to replace the device home screen **MUST** declare the platform feature `android.software.home_screen`.

3.8.2. Widgets

Widgets are optional for all Android device implementations, but **SHOULD** be supported on Android Handheld devices.

Android defines a component type and corresponding API and lifecycle that allows applications to expose an “AppWidget” to the end user [[Resources, 24](#)] a feature that is **STRONGLY RECOMMENDED** to be supported on Handheld Device implementations. Device implementations that support embedding widgets on the home screen **MUST** meet the following requirements and declare support for platform feature `android.software.app_widgets`.

- Device launchers **MUST** include built-in support for AppWidgets, and expose user interface affordances to add, configure, view, and remove AppWidgets directly within the Launcher.
- Device implementations **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 [[Resources, 24](#)] for details.
- Device implementations that include support for lock screen **MAY** support application widgets on the lock screen.

3.8.3. Notifications

Android includes APIs that allow developers to notify users of notable events [[Resources, 25](#)], using hardware and software features of the device.

Some APIs allow applications to perform notifications or attract attention using hardware—specifically sound, vibration, and light. Device implementations **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](#).

Additionally, the implementation **MUST** correctly render all resources (icons, animation files etc.) provided for in the APIs [[Resources, 26](#)], or in the Status/System Bar icon style guide [[Resources, 27](#)], which in the case of an Android Television device includes the possibility to not display the notifications. Device implementers **MAY** provide an alternative user experience for notifications than that provided by the reference Android Open Source implementation; however, such alternative notification systems **MUST** support existing notification resources, as above.

Android includes support for various notifications, such as:

- **Rich notifications.** Interactive Views for ongoing notifications.
- **Heads-up notifications.** Interactive Views users can act on or dismiss without leaving the current app.
- **Lockscreen notifications.** Notifications shown over a lock screen with granular control on visibility.

Android device implementations, when such notifications are made visible, **MUST** properly execute Rich and Heads-up notifications and include the title/name, icon, text as documented in the Android APIs [[Resources, 28](#)].

Android includes Notification Listener Service 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 **MUST** correctly and promptly send notifications in their entirety to all such installed and user-enabled listener services, including any and all metadata attached to the Notification object.

3.8.4. Search

Android includes APIs [[Resources, 29](#)] 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. Device implementations **SHOULD** implement the APIs that allow developers to reuse this user interface to provide search within their own applications. Device implementations that implement the global search interface **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 this functionality, the default behavior **SHOULD** be to display web search engine results and suggestions.

Android device implementations **SHOULD** implement an assistant on the device to handle the Assist action [[Resources, 30](#)].

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 [[Resources, 31](#)]. Device implementations supporting the Assist action **MUST** indicate clearly to the end user when the the context is shared by displaying a white light around the edges of the screen. To ensure clear visibility to the end user, the indication **MUST** meet or exceed the duration and brightness of the Android Open Source Project implementation.

3.8.5. 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 [[Resources, 32](#)]. Device implementations **MUST** display Toasts from

applications to end users in some high-visibility 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” 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 [[Resources, 33](#)]. Device implementations MUST NOT alter any of the Holo theme attributes exposed to applications [[Resources, 34](#)].

Android includes a “Material” theme family as a set of defined styles for application developers to use if they want to match the design theme’s look and feel across the wide variety of different Android device types. Device implementations MUST support the “Material” theme family and MUST NOT alter any of the Material theme attributes or their assets exposed to applications [[Resources, 35](#)].

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 [[Resources, 34](#)].

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. Therefore, Android device implementations 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. When an app requests a light status bar, Android device implementations MUST change the color of the system status icons to black [[Resources, 34](#)].

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 [[Resources, 36](#)]. 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, and when implemented MUST report the platform feature flag `android.software.live_wallpaper`.

3.8.8. Activity Switching

As the Recent function navigation key is OPTIONAL, the requirements to implement the overview screen is OPTIONAL for Android Television devices and Android Watch devices.

The upstream Android source code includes the overview screen [[Resources, 37](#)], 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 but MUST meet the following requirements:

- MUST display affiliated recents as a group that moves together.
- MUST support at least up to 20 displayed activities.
- MUST at least display the title of 4 activities at a time.
- SHOULD display highlight color, icon, screen title in recents.
- MUST implement the screen pinning behavior [[Resources, 38](#)] and provide the user with a settings menu to toggle the feature.
- SHOULD display a closing affordance ("x") but MAY delay this until user interacts with screens.

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 [[Resources, 39](#)]. Device implementations that allow users to use third-party input methods on the device MUST declare the platform feature `android.software.input_methods` and support IME APIs as defined in the Android SDK documentation.

Device implementations that declare the `android.software.input_methods` feature MUST provide a user-accessible mechanism to add and configure third-party input methods. Device implementations MUST display the settings interface in response to the `android.settings.INPUT_METHOD_SETTINGS` intent.

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 [[Resources, 40](#)]. Device implementations that support a lock screen, unless an Android Automotive or Watch implementation, MUST display the Lockscreen Notifications including the Media Notification Template.

3.8.11. Dreams

Android includes support for interactive screensavers called Dreams [[Resources, 41](#)]. Dreams allows 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 Dreams, but other types of device implementations SHOULD include support for Dreams and provide a settings option for users to configure Dreams in response to the `android.settings.DREAM_SETTINGS` intent.

3.8.12. Location

When a device has a hardware sensor (e.g. GPS) that is capable of providing the location coordinates, location modes MUST be displayed in the Location menu within Settings [[Resources, 42](#)].

3.8.13. Unicode and Font

Android includes support for color emoji characters. When Android device implementations include an IME, devices SHOULD provide an input method to the user for the Emoji characters defined in Unicode 6.1 [[Resources, 43](#)]. All devices MUST be capable of rendering these emoji characters in color glyph.

Android includes 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—which MUST all be included for the languages available on the device and 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.

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 [[Resources, 44](#)]. Device implementations MUST provide an implementation of the DevicePolicyManager class [[Resources, 45](#)]. Device implementations that include support for PIN (numeric) or PASSWORD (alphanumeric) based lock screens MUST support the full range of device administration policies defined in the Android SDK documentation [[Resources, 44](#)] and report the platform feature `android.software.device_admin`.

3.9.1 Device Provisioning

3.9.1.1 Device owner provisioning

If a device implementation declares the `android.software.device_admin` feature, the out of box setup flow MUST make it possible to enroll a Device Policy Controller (DPC) application as the Device Owner app [[Resources, 46](#)]. Device implementations MAY have a preinstalled application performing device administration functions but this application MUST NOT be set as the Device Owner app without explicit consent or action from the user or the administrator of the device.

The device owner provisioning process (the flow initiated by `android.app.action.PROVISION_MANAGED_DEVICE` [[Resources, 47](#)]) user experience MUST align with the AOSP implementation

If the device implementation reports `android.hardware.nfc`, it MUST have NFC enabled, even during the out-of-box setup flow, in order to allow for NFC provisioning of Device owners [[Resources, 48](#)].

3.9.1.2 Managed profile provisioning

If a device implementation declares the `android.software.managed_users`, it MUST be possible to enroll a Device Policy Controller (DPC) application as the owner of a new Managed Profile [[Resources, 49](#)]

The managed profile provisioning process (the flow initiated by `android.app.action.PROVISION_MANAGED_PROFILE` [[Resources, 50](#)]) user experience MUST align with the AOSP implementation

3.9.2 Managed Profile Support

Managed profile capable devices are those devices that:

- Declare `android.software.device_admin` (see [section 3.9 Device Administration](#))
- Are not low RAM devices (see [section 7.6.1](#))
- Allocate internal (non-removable) storage as shared storage (see [section 7.6.2](#))

Managed profile capable devices MUST:

- Declare the platform feature flag `android.software.managed_users`.
- Support managed profiles via the `android.app.admin.DevicePolicyManager` APIs

- Allow one and only one managed profile to be created [[Resources, 50](#)]
- 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
- Display a notification icon (similar to the AOSP upstream work badge) to indicate when user is within a managed profile application
- 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
- Where a managed profile exists, 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
- Where a managed profile exists, 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 user or managed profile.
 - Independent management of accounts within the primary user or managed profile.
- Ensure the default dialer can look up caller information from the managed profile (if one exists) alongside those from the primary profile, if the Device Policy Controller permits it.
- 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.

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 [[Resources, 51](#)].

Device implementations include the following requirements:

- Android Automotive implementations SHOULD provide an implementation of the Android accessibility framework consistent with the default Android implementation.
- Device implementations (Android Automotive excluded) MUST provide an implementation of the Android accessibility framework consistent with the default Android implementation.
- Device implementations (Android Automotive excluded) MUST support third-party accessibility service implementations through the android.accessibilityservice APIs [[Resources, 52](#)]
- Device implementations (Android Automotive excluded) MUST generate AccessibilityEvents and deliver these events to all registered AccessibilityService implementations in a manner consistent with the default Android implementation
- Device implementations (Android Automotive and Android Watch devices with no audio output excluded), MUST provide a user-accessible mechanism to enable and disable accessibility services, and MUST display this interface in response to the android.provider.Settings.ACTION_ACCESSIBILITY_SETTINGS intent.

Additionally, device implementations SHOULD provide an implementation of an accessibility service on the device, and SHOULD provide a mechanism for users to enable the accessibility service during

device setup. An open source implementation of an accessibility service is available from the Eyes Free project [[Resources, 53](#)].

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 [[Resources, 54](#)]. Device implementations reporting the feature `android.hardware.audio.output` MUST meet these requirements related to the Android TTS framework.

Android Automotive implementations:

- MUST support the Android TTS framework APIs.
- MAY support installation of third-party TTS engines. If supported, partners MUST provide a user-accessible interface that allows the user to select a TTS engine for use at system level.

All other device implementations:

- MUST support the Android TTS framework APIs and SHOULD include a TTS engine supporting the languages available on the device. Note that the upstream Android open source software includes a full-featured TTS engine implementation.
- MUST support installation of third-party TTS engines
- MUST provide a user-accessible interface that allows users to select a TTS engine for use at the 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. Android Television device implementations MUST support TV Input Framework [[Resources, 55](#)].

Device implementations that support TIF MUST declare the platform feature `android.software.live_tv`.

3.12.1. TV App

Any device implementation that declares support for Live TV MUST have an installed TV application (TV App). The Android Open Source Project provides an implementation of the TV App.

The TV App MUST provide facilities to install and use TV Channels [[Resources, 56](#)] and meet the following requirements:

- Device implementations MUST allow third-party TIF-based inputs (third-party inputs) [[Resources, 57](#)] to be installed and managed.
- Device implementations MAY provide visual separation between pre-installed TIF-based inputs (installed inputs) [[Resources, 58](#)] and third-party inputs.
- The device implementations MUST 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).

3.12.1.1. Electronic Program Guide

Android Television device implementations MUST show an informational and interactive overlay, which MUST include an electronic program guide (EPG) generated from the values in the

TvContract.Programs fields [[Resources, 59](#)]. The EPG MUST meet the following requirements:

- The EPG MUST display information from all installed inputs and third-party inputs.
- The EPG MAY provide visual separation between the installed inputs and third-party inputs.
- The EPG is STRONGLY RECOMMENDED to display installed inputs and third-party inputs with equal prominence. The EPG MUST NOT display the third-party inputs more than a single navigation action away from the installed inputs on the EPG.
- On channel change, device implementations MUST display EPG data for the currently playing program.

3.12.1.2. Navigation

Android Television device input devices (i.e. remote control, remote control application, or game controller) MUST allow navigation to all actionable sections of the screen via the D-pad. D-pad up and down MUST be used to change live TV channels when there is no actionable section on the screen.

The TV App SHOULD pass key events to HDMI inputs through CEC.

3.12.1.3. TV input app linking

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) [[Resources, 60](#)]. The TV App MUST show TV input app linking when it is provided.

4. Application Packaging Compatibility

Device implementations MUST install and run Android “.apk” files as generated by the “aapt” tool included in the official Android SDK [[Resources, 61](#)].

Devices implementations MUST NOT extend either the .apk [[Resources, 62](#)], Android Manifest [[Resources, 49](#)], Dalvik bytecode [[Resources, 23](#)], or RenderScript bytecode formats in such a way that would prevent those files from installing and running correctly on other compatible devices.

5. Multimedia Compatibility

5.1. Media Codecs

Device implementations MUST support the core media formats specified in the Android SDK documentation [[Resources, 64](#)] except where explicitly permitted in this document. Specifically, device implementations MUST support the media formats, encoders, decoders, file types, and container formats defined in the tables below and reported via MediaCodecList [[Resources, 65](#)]. Device implementations MUST also be able to decode all profiles reported in its CamcorderProfile [[Resources, 66](#)] and MUST be able to decode all formats it can encode. All of these codecs 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.

5.1.1. Audio Codecs

--	--	--	--	--

Format/Codec	Encoder	Decoder	Details	Supported File Types/Container Formats
MPEG-4 AAC Profile (AAC LC)	REQUIRED ¹	REQUIRED	Support for mono/stereo/5.0/5.1 ² content with standard sampling rates from 8 to 48 kHz.	<ul style="list-style-type: none"> • 3GPP (.3gp) • MPEG-4 (.mp4, .m4a) • ADTS raw AAC (.aac, decode in Android 3.1+, encode in Android 4.0+, ADIF not supported) • MPEG-TS (.ts, not seekable, Android 3.0+)
MPEG-4 HE AAC Profile (AAC+)	REQUIRED ¹ (Android 4.1+)	REQUIRED	Support for mono/stereo/5.0/5.1 ² content with standard sampling rates from 16 to 48 kHz.	
MPEG-4 HE AACv2 Profile (enhanced AAC+)		REQUIRED	Support for mono/stereo/5.0/5.1 ² content with standard sampling rates from 16 to 48 kHz.	
AAC ELD (enhanced low delay AAC)	REQUIRED ¹ (Android 4.1+)	REQUIRED (Android 4.1+)	Support for mono/stereo content with standard sampling rates from 16 to 48 kHz.	
AMR-NB	REQUIRED ³	REQUIRED ³	4.75 to 12.2 kbps sampled @ 8 kHz	3GPP (.3gp)
AMR-WB	REQUIRED ³	REQUIRED ³	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16 kHz	
FLAC		REQUIRED (Android 3.1+)	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		REQUIRED	Mono/Stereo 8-320Kbps constant (CBR) or variable bitrate (VBR)	MP3 (.mp3)
MIDI		REQUIRED	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	<ul style="list-style-type: none"> • Type 0 and 1 (.mid, .xmf, .mxmf) • RTTTL/RTX (.rtttl, .rtx) • OTA (.ota)

				<ul style="list-style-type: none"> • iMelody (.imy)
Vorbis		REQUIRED		<ul style="list-style-type: none"> • Ogg (.ogg) • Matroska (.mkv, Android 4.0+)
PCM/WAVE	REQUIRED ⁴ (Android 4.1+)	REQUIRED	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		REQUIRED (Android 5.0+)		Matroska (.mkv)

1 Required for device implementations that define android.hardware.microphone but optional for Android Watch device implementations.

2 Only downmix of 5.0/5.1 content is required; recording or rendering more than 2 channels is optional.

3 Required for Android Handheld device implementations.

4 Required for device implementations that define android.hardware.microphone, including Android Watch device implementations.

5.1.2. Image Codecs

Format/Codec	Encoder	Decoder	Details	Supported File Types/Container Formats
JPEG	REQUIRED	REQUIRED	Base+progressive	JPEG (.jpg)
GIF		REQUIRED		GIF (.gif)
PNG	REQUIRED	REQUIRED		PNG (.png)
BMP		REQUIRED		BMP (.bmp)
WebP	REQUIRED	REQUIRED		WebP (.webp)

5.1.3. Video Codecs

Format/Codec	Encoder	Decoder	Details	Supported File Types/Container Formats
H.263	REQUIRED ¹	REQUIRED ²		<ul style="list-style-type: none"> • 3GPP (.3gp) • MPEG-4 (.mp4)
H.264 AVC	REQUIRED ²	REQUIRED ²	See section 5.2 and 5.3 for details	<ul style="list-style-type: none"> • 3GPP (.3gp) • MPEG-4 (.mp4) • MPEG-2 TS (.ts, AAC audio only, not seekable, Android 3.0+)

H.265 HEVC		REQUIRED ⁵	See section 5.3 for details	MPEG-4 (.mp4)
MPEG-2		STRONGLY RECOMMENDED ⁶	Main Profile	MPEG2-TS
MPEG-4 SP		REQUIRED ²		3GPP (.3gp)
VP8 ³	REQUIRED ² (Android 4.3+)	REQUIRED ² (Android 2.3.3+)	See section 5.2 and 5.3 for details	<ul style="list-style-type: none"> • WebM (.webm) [Resources, 67] • Matroska (.mkv, Android 4.0+)⁴
VP9		REQUIRED ² (Android 4.4+)	See section 5.3 for details	<ul style="list-style-type: none"> • WebM (.webm) [Resources, 67] • Matroska (.mkv, Android 4.0+)⁴

1 Required for device implementations that include camera hardware and define android.hardware.camera or android.hardware.camera.front.

2 Required for device implementations except Android Watch devices.

3 For acceptable quality of web video streaming and video-conference services, device implementations SHOULD use a hardware VP8 codec that meets the requirements in [[Resources, 68](#)].

4 Device implementations SHOULD support writing Matroska WebM files.

5 STRONGLY RECOMMENDED for Android Automotive, optional for Android Watch, and required for all other device types.

6 Applies only to Android Television device implementations.

5.2. Video Encoding

Video codecs are optional for Android Watch device implementations.

Android device implementations with H.263 encoders, MUST support Baseline Profile Level 45.

Android device implementations with H.264 codec support, MUST support Baseline Profile Level 3 and the following SD (Standard Definition) video encoding profiles and SHOULD support Main Profile Level 4 and the following HD (High Definition) video encoding profiles. Android Television devices are STRONGLY RECOMMENDED to encode HD 1080p video at 30 fps.

	SD (Low quality)	SD (High quality)	HD 720p ¹	HD 1080p ¹
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

1 When supported by hardware, but STRONGLY RECOMMENDED for Android Television devices.

Android device implementations with VP8 codec support MUST support the SD video encoding profiles and SHOULD support the following HD (High Definition) video encoding profiles.

	SD (Low quality)	SD (High quality)	HD 720p ¹	HD 1080p ¹
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

1 When supported by hardware.

5.3. Video Decoding

Video codecs are optional for Android Watch device implementations.

Device implementations 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 on the device.

Android device implementations with H.263 decoders, MUST support Baseline Profile Level 30.

Android device implementations with MPEG-4 decoders, MUST support Simple Profile Level 3.

Android device implementations with H.264 decoders, MUST support Main Profile Level 3.1 and the following SD video decoding profiles and SHOULD support the HD decoding profiles. Android Television devices MUST support High Profile Level 4.2 and the HD 1080p decoding profile.

	SD (Low quality)	SD (High quality)	HD 720p ¹	HD 1080p ¹
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 ²
Video bitrate	800 Kbps	2 Mbps	8 Mbps	20 Mbps

1 REQUIRED for when the height as reported by the `Display.getSupportedModes()` method is not smaller than the video resolution.

2 REQUIRED for Android Television device implementations.

Android device implementations when supporting VP8 codec as described in [section 5.1.3](#), MUST support the following SD decoding profiles and SHOULD support the HD decoding profiles. Android Television devices MUST support the HD 1080p decoding profile.

	SD (Low quality)	SD (High quality)	HD 720p ¹	HD 1080p ¹
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 ²	30 / 60 fps ²
Video bitrate	800 Kbps	2 Mbps	8 Mbps	20 Mbps

1 REQUIRED for when the height as reported by the `Display.getSupportedModes()` method is not smaller than the video resolution.

2 REQUIRED for Android Television device implementations.

Android device implementations, when supporting VP9 codec as described in [section 5.1.3](#), MUST support the following SD video decoding profiles and SHOULD support the HD decoding profiles. Android Television devices are STRONGLY RECOMMENDED to support the HD 1080p decoding profile and SHOULD support the UHD decoding profile. When the UHD video decoding profile is supported, it MUST support 8-bit color depth and SHOULD support VP9 Profile 2 (10-bit).

	SD (Low quality)	SD (High quality)	HD 720p ¹	HD 1080p ²	UHD ²
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	60 fps	60 fps

Video bitrate 600 Kbps | 1.6 Mbps | 4 Mbps | 10 Mbps | 20 Mbps

1 Required for Android Television device implementations, but for other type of devices only when supported by hardware.

2 **STRONGLY RECOMMENDED** for existing Android Television device implementations when supported by hardware.

Android device implementations, when supporting H.265 codec as described in [section 5.1.3](#), **MUST** support the Main Profile Level 3 Main tier and the following SD video decoding profiles and **SHOULD** support the HD decoding profiles. Android Television devices **MUST** support the Main Profile Level 4.1 Main tier and the HD 1080p decoding profile and **SHOULD** support Main10 Level 5 Main Tier profile and the UHD decoding profile.

	SD (Low quality)	SD (High quality)	HD 720p ¹	HD 1080p ¹	UHD ²
Video resolution	352 x 288 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	60 fps ²	60 fps
Video bitrate	600 Kbps	1.6 Mbps	4 Mbps	10 Mbps	20 Mbps

1 Required for Android Television device implementations, but for other type of devices only when supported by hardware.

2 **STRONGLY RECOMMENDED** for existing Android Television device implementations when supported by hardware.

5.4. Audio Recording

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

5.4.1. Raw Audio Capture

Device implementations that declare `android.hardware.microphone` **MUST** allow capture of raw audio content with the following characteristics:

- **Format:** Linear PCM, 16-bit
- **Sampling rates:** 8000, 11025, 16000, 44100
- **Channels:** Mono

The capture for the above sample rates **MUST** be done without up-sampling, and any down-sampling **MUST** include an appropriate anti-aliasing filter.

Device implementations that declare `android.hardware.microphone` **SHOULD** allow capture of raw audio content with the following characteristics:

- **Format:** Linear PCM, 16-bit
- **Sampling rates:** 22050, 48000
- **Channels:** Stereo

If capture for the above sample rates is supported, then the capture **MUST** be done without up-sampling at any ratio higher than 16000:22050 or 44100:48000. Any up-sampling or down-sampling **MUST** include an appropriate anti-aliasing filter.

5.4.2. Capture for Voice Recognition

In addition to the above recording specifications, when an application has started recording an audio stream using the `android.media.MediaRecorder.AudioSource.VOICE_RECOGNITION` audio source:

- The device SHOULD exhibit approximately flat amplitude versus frequency characteristics: specifically, ± 3 dB, from 100 Hz to 4000 Hz.
- Audio input sensitivity SHOULD be set such that a 90 dB sound power level (SPL) source at 1000 Hz yields RMS of 2500 for 16-bit samples.
- PCM amplitude levels SHOULD 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.
- Total harmonic distortion SHOULD be less than 1% for 1 kHz at 90 dB SPL input level at the microphone.
- Noise reduction processing, if present, MUST be disabled.
- Automatic gain control, if present, MUST be disabled

If the platform supports noise suppression technologies tuned for speech recognition, the effect MUST be controllable from the `android.media.audiofx.NoiseSuppressor` API. Moreover, the `UUID` field for the noise suppressor's effect descriptor MUST uniquely identify each implementation of the noise suppression technology.

5.4.3. Capture for Rerouting of Playback

The `android.media.MediaRecorder.AudioSource` class includes the `REMOTE_SUBMIX` audio source. Devices that declare `android.hardware.audio.output` 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 can capture a mix of all audio streams except for the following:

- `STREAM_RING`
- `STREAM_ALARM`
- `STREAM_NOTIFICATION`

5.5. Audio Playback

Device implementations that declare `android.hardware.audio.output` MUST conform to the requirements in this section.

5.5.1. Raw Audio Playback

The device 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

The device 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 [[Resources, 69](#)]. Device

implementations that declare the feature `android.hardware.audio.output`:

- MUST support the `EFFECT_TYPE_EQUALIZER` and `EFFECT_TYPE_LOUDNESS_ENHANCER` implementations controllable through the `AudioEffect` subclasses `Equalizer`, `LoudnessEnhancer`.
- 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

Android Television device implementations 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).

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 can be heard by an external listener or observed by a transducer.
- **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 an external sound is presented to the device and when an application reads the corresponding frame of PCM-coded data.
- **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 variance among separate measurements of cold output latency values.
- **cold input jitter.** The variance 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 term allows processing time for the app and 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; see [NDK_root/docs/opensles/index.html](https://source.android.com/docs/ndk/docs/opensles/index.html).

Device implementations that declare `android.hardware.audio.output` are **STRONGLY RECOMMENDED** to meet or exceed these audio output requirements:

- cold output latency of 100 milliseconds or less
- continuous output latency of 45 milliseconds or less
- minimize the cold output jitter

If a device implementation meets the requirements of this section 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, it is **STRONGLY RECOMMENDED** to report support for low-latency audio, by reporting the feature `android.hardware.audio.low_latency` via the `android.content.pm.PackageManager` class [[Resources, 70](#)]. Conversely, if the device implementation does not meet these requirements it **MUST NOT** report support for low-latency audio.

Device implementations that include `android.hardware.microphone` are **STRONGLY RECOMMENDED** to meet these input audio requirements:

- cold input latency of 100 milliseconds or less
- continuous input latency of 30 milliseconds or less
- continuous round-trip latency of 50 milliseconds or less
- minimize the cold input jitter

5.7. Network Protocols

Devices **MUST** support the media network protocols for audio and video playback as specified in the Android SDK documentation [[Resources, 64](#)]. Specifically, devices **MUST** support the following media network protocols:

- RTSP (RTP, SDP)
- HTTP(S) progressive streaming
- HTTP(S) Live Streaming draft protocol, Version 3 [[Resources, 71](#)]

5.8. Secure Media

Device implementations that support secure video output and are capable of supporting secure surfaces **MUST** declare support for `Display.FLAG_SECURE`. Device implementations that declare support for `Display.FLAG_SECURE`, if they support a wireless display protocol, **MUST** secure the link with a cryptographically strong mechanism such as HDCP 2.x or higher for Miracast wireless displays. Similarly if they support a wired external display, the device implementations **MUST** support HDCP 1.2 or higher. Android Television device implementations **MUST** support HDCP 2.2 for devices supporting 4K resolution and HDCP 1.4 or above for lower resolutions. The upstream Android open source implementation includes support for wireless (Miracast) and wired (HDMI) displays that satisfies this requirement.

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 **STRONGLY RECOMMENDED** to report support for feature `android.software.midi` via the `android.content.pm.PackageManager` class [[Resources, 70](#)].

The MIDI-capable hardware transports are:

- USB host mode (section 7.7 USB)
- USB peripheral mode (section 7.7 USB)

Conversely, 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 **MUST NOT** report support for feature `android.software.midi`.

MIDI over Bluetooth LE acting in central role (section 7.4.3 Bluetooth) is in trial use status. A device implementation that reports feature `android.software.midi`, and which provides generic non-MIDI

connectivity over Bluetooth LE, SHOULD support MIDI over Bluetooth LE.

5.10. Professional Audio

If a device implementation meets *all* of the following requirements, it is STRONGLY RECOMMENDED to report support for feature `android.hardware.audio.pro` via the `android.content.pm.PackageManager` class [[Resources, 70](#)].

- The device implementation MUST report support for feature `android.hardware.audio.low_latency`.
- 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.
- If the device includes a 4 conductor 3.5mm audio jack, the continuous round-trip audio latency MUST be 20 milliseconds or less over the audio jack path, and SHOULD be 10 milliseconds or less over at the audio jack path.
- The device implementation MUST include a USB port(s) supporting USB host mode and USB peripheral mode.
- The USB host mode MUST implement the USB audio class.
- If the device includes an HDMI port, the device implementation MUST support output in stereo and eight channels at 20-bit or 24-bit depth and 192 kHz without bit-depth loss or resampling.
- The device implementation MUST report support for feature `android.software.midi`.
- If the device includes a 4 conductor 3.5mm audio jack, the device implementation is STRONGLY RECOMMENDED to comply with section [Mobile device \(jack\) specifications](#) of the [Wired Audio Headset Specification \(v1.1\)](#).

6. Developer Tools and Options Compatibility

6.1. Developer Tools

Device implementations MUST support the Android Developer Tools provided in the Android SDK. Android compatible devices MUST be compatible with:

- **Android Debug Bridge (adb)** [[Resources, 72](#)]

Device implementations MUST support all adb functions as documented in the Android SDK including `dumpsys` [[Resources, 73](#)]. The device-side adb daemon MUST be inactive by default and there MUST be a user-accessible mechanism to turn on the Android Debug Bridge. If a device implementation omits USB peripheral mode, it MUST implement the Android Debug Bridge via local-area network (such as Ethernet or 802.11).

Android includes support for secure adb. Secure adb enables adb on known authenticated hosts. Device implementations MUST support secure adb.

- **Dalvik Debug Monitor Service (ddms)** [[Resources, 74](#)]

Device implementations 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** [[Resources, 75](#)]

Device implementations MUST include the Monkey framework, and make it available for applications

to use.

- **SysTrace** [[Resources, 76](#)]

Device implementations 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.

Most Linux-based systems and Apple Macintosh systems recognize Android devices using the standard Android SDK tools, without additional support; however Microsoft Windows systems typically require a driver for new Android devices. (For instance, new vendor IDs and sometimes new device IDs require custom USB drivers for Windows systems.) If a device implementation is unrecognized by the adb tool as provided in the standard Android SDK, device implementers MUST provide Windows drivers allowing developers to connect to the device using the adb protocol. These drivers MUST be provided for Windows XP, Windows Vista, Windows 7, Windows 8 and Windows 10 in both 32-bit and 64-bit versions.

6.2. Developer Options

Android includes support for developers to configure application development-related settings. Device implementations MUST honor the `android.settings.APPLICATION_DEVELOPMENT_SETTINGS` intent to show application development-related settings [[Resources, 77](#)]. 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. Device implementations MUST provide a consistent experience for Developer Options. Specifically, device implementations MUST hide Developer Options by default and MUST provide a mechanism to enable Developer Options that is consistent with the upstream Android implementation.

7. Hardware Compatibility

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

- Complete class definitions (as documented by the SDK) for the component APIs MUST still be presented.
- The API's behaviors MUST be implemented as no-ops in some reasonable fashion.
- API methods MUST return null values where permitted by the SDK documentation.
- API methods MUST return no-op implementations of classes where null values are not permitted by the SDK documentation.
- API methods MUST NOT throw exceptions not documented by the SDK documentation.

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

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. [[Resources, 70](#)]

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 [[Resources, 78](#)]. 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: $\text{pixels} = \text{dps} * (\text{density}/160)$.

7.1.1. Screen Configuration

7.1.1.1. Screen Size

Android Watch devices (detailed in [section 2](#)) MAY have smaller screen sizes as described in this section.

The Android UI framework supports a variety of different screen sizes, and allows applications to query the device screen size (aka "screen layout") via `android.content.res.Configuration.screenLayout` with the `SCREENLAYOUT_SIZE_MASK`. Device implementations MUST report the correct screen size as defined in the Android SDK documentation [[Resources, 78](#)] and determined by the upstream Android platform. Specifically, device implementations MUST report the correct screen size according to the following logical density-independent pixel (dp) screen dimensions.

- Devices MUST have screen sizes of at least 426 dp x 320 dp ('small'), unless it is an Android Watch device.
- Devices that report screen size 'normal' MUST have screen sizes of at least 480 dp x 320 dp.
- Devices that report screen size 'large' MUST have screen sizes of at least 640 dp x 480 dp.
- Devices that report screen size 'xlarge' MUST have screen sizes of at least 960 dp x 720 dp.

In addition,

- Android Watch devices MUST have a screen with the physical diagonal size in the range from 1.1 to 2.5 inches.
- Other types of Android device implementations, with a physically integrated screen, MUST have a screen at least 2.5 inches in physical diagonal size.

Devices MUST NOT change their reported screen size at any time.

Applications optionally indicate which screen sizes they support via the `<supports-screens>` attribute in the `AndroidManifest.xml` file. Device implementations MUST correctly honor applications' stated support for small, normal, large, and xlarge screens, as described in the Android SDK documentation.

7.1.1.2. Screen Aspect Ratio

Android Watch devices MAY have an aspect ratio of 1.0 (1:1).

The screen aspect ratio MUST be a value from 1.3333 (4:3) to 1.86 (roughly 16:9), but Android Watch devices MAY have an aspect ratio of 1.0 (1:1) because such a device implementation will use a

UI_MODE_TYPE_WATCH as the `android.content.res.Configuration.uiMode`.

7.1.1.3. Screen Density

The Android UI framework defines a set of standard logical densities to help application developers target application resources. Device implementations **MUST** report only one of the following logical Android framework densities through the `android.util.DisplayMetrics` APIs, and **MUST** execute applications at this standard density and **MUST NOT** change the value at any time for the default display.

- 120 dpi (ldpi)
- 160 dpi (mdpi)
- 213 dpi (tvdpi)
- 240 dpi (hdpi)
- 280 dpi (280dpi)
- 320 dpi (xhdpi)
- 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.

7.1.2. Display Metrics

Device implementations **MUST** report correct values for all display metrics defined in `android.util.DisplayMetrics` [[Resources, 79](#)] and **MUST** report the same values regardless of whether the embedded or external screen is used as the default display.

7.1.3. Screen Orientation

Devices **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`.

Devices that report both screen orientations **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. Device implementations **MAY** select either portrait or landscape orientation as the default.

Devices **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.

Devices **MUST NOT** change the reported screen size or density when changing orientation.

7.1.4. 2D and 3D Graphics Acceleration

Device implementations MUST support both OpenGL ES 1.0 and 2.0, as embodied and detailed in the Android SDK documentations. Device implementations SHOULD support OpenGL ES 3.0 or 3.1 on devices capable of supporting it. Device implementations MUST also support Android RenderScript, as detailed in the Android SDK documentation [[Resources, 80](#)].

Device implementations MUST also correctly identify themselves as supporting OpenGL ES 1.0, OpenGL ES 2.0, OpenGL ES 3.0 or OpenGL 3.1. That is:

- The managed APIs (such as via the `GLLES10.getString()` method) MUST report support for OpenGL ES 1.0 and OpenGL ES 2.0.
- The native C/C++ OpenGL APIs (APIs available to apps via `libGLES_v1CM.so`, `libGLES_v2.so`, or `libEGL.so`) MUST report support for OpenGL ES 1.0 and OpenGL ES 2.0.
- Device implementations that declare support for OpenGL ES 3.0 or 3.1 MUST support the corresponding managed APIs and include support for native C/C++ APIs. On device implementations that declare support for OpenGL ES 3.0 or 3.1, `libGLESv2.so` MUST export the corresponding function symbols in addition to the OpenGL ES 2.0 function symbols.

In addition to OpenGL ES 3.1, Android provides an extension pack with Java interfaces [[Resources, 81](#)] and native support for advanced graphics functionality such as tessellation and the ASTC texture compression format. Android device implementations MAY support this extension pack, and—only if fully implemented—MUST identify the support through the `android.hardware.opengles.aep` feature flag.

Also, device implementations MAY implement any desired OpenGL ES extensions. However, device implementations MUST report via the OpenGL ES managed and native APIs all extension strings that they do support, and conversely MUST NOT report extension strings that they do not support.

Note that Android includes support for applications to optionally specify that they require specific OpenGL texture compression formats. These formats are typically vendor-specific. Device implementations are not required by Android to implement any specific texture compression format. However, they SHOULD accurately report any texture compression formats that they do support, via the `getString()` method in the OpenGL API.

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 [[Resources, 82](#)].

Device implementations 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.

In addition, device implementations MUST exhibit behavior consistent with the Android SDK documentation on hardware acceleration [[Resources, 82](#)].

Android includes a `TextureView` object that lets developers directly integrate hardware-accelerated OpenGL ES textures as rendering targets in a UI hierarchy. Device implementations MUST support the `TextureView` API, and MUST exhibit consistent behavior with the upstream Android implementation.

Android includes support for `EGL_ANDROID_RECORDABLE`, an `EGLConfig` attribute that indicates whether the `EGLConfig` supports rendering to an `ANativeWindow` that records images to a video. Device implementations MUST support `EGL_ANDROID_RECORDABLE` extension [[Resources, 83](#)].

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.

- Android Automotive does not support legacy compatibility mode.
- All other device implementations **MUST** include support for legacy application compatibility mode as implemented by the upstream Android open source code. That is, device implementations **MUST NOT** alter the triggers or thresholds at which compatibility mode is activated, and **MUST NOT** alter the behavior of the compatibility mode itself.

7.1.6. 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.

- Devices **MUST** support displays capable of rendering 16-bit color graphics and **SHOULD** support displays capable of 24-bit color graphics.
- Devices **MUST** support displays capable of rendering animations.
- The display technology used **MUST** have a pixel aspect ratio (PAR) between 0.9 and 1.15. That is, the pixel aspect ratio **MUST** be near square (1.0) with a 10 ~ 15% tolerance.

7.1.7. Secondary Displays

Android includes support for secondary display to enable media sharing capabilities and developer APIs for accessing external displays. If a device supports an external display either via a wired, wireless, or an embedded additional display connection then the device implementation **MUST** implement the display manager API as described in the Android SDK documentation [[Resources, 84](#)].

7.2. Input Devices

Devices **MUST** support a touchscreen or meet the requirements listed in 7.2.2 for non-touch navigation.

7.2.1. Keyboard

Android Watch and Android Automotive implementations **MAY** implement a soft keyboard. All other device implementations **MUST** implement a soft keyboard and:

Device implementations:

- **MUST** include support for the Input Management Framework (which allows third-party developers to create Input Method Editors—i.e. soft keyboard) as detailed at <http://developer.android.com>.
- **MUST** provide at least one soft keyboard implementation (regardless of whether a hard keyboard is present) except for Android Watch devices where the screen size makes it less reasonable to have a soft keyboard.
- **MAY** include additional soft keyboard implementations.
- **MAY** include a hardware keyboard.
- **MUST NOT** include a hardware keyboard that does not match one of the formats specified in `android.content.res.Configuration.keyboard` [[Resources, 85](#)] (QWERTY or 12-key).

7.2.2. Non-touch Navigation

Android Television devices **MUST** support D-pad.

Device implementations:

- MAY omit a non-touch navigation option (trackball, d-pad, or wheel) if the device implementation is not an Android Television device.
- MUST report the correct value for `android.content.res.Configuration.navigation` [[Resources, 85](#)].
- 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 availability and visibility requirement of the Home, Recents, and Back functions differ between device types as described in this section.

The Home, Recents, and Back functions (mapped to the key events `KEYCODE_HOME`, `KEYCODE_APP_SWITCH`, `KEYCODE_BACK`, respectively) are essential to the Android navigation paradigm and therefore:

- Android Handheld device implementations MUST provide the Home, Recents, and Back functions.
- Android Television device implementations MUST provide the Home and Back functions.
- Android Watch device implementations MUST have the Home function available to the user, and the Back function except for when it is in `UI_MODE_TYPE_WATCH`.
- Android Automotive implementations MUST provide the Home function and MAY provide Back and Recent functions.
- All other types of device implementations MUST provide the Home and Back functions.

These functions MAY be implemented via dedicated physical buttons (such as mechanical or capacitive touch buttons), or MAY be implemented using dedicated software keys on a distinct portion of the screen, gestures, touch panel, etc. Android supports both implementations. All of these functions MUST be accessible with a single action (e.g. tap, double-click or gesture) when visible.

Recents function, if provided, MUST have a visible button or icon unless hidden together with other navigation functions in full-screen mode. This does not apply to devices upgrading from earlier Android versions that have physical buttons for navigation and no recents key.

The Home and Back functions, if provided, MUST each have a visible button or icon unless hidden together with other navigation functions in full-screen mode or when the `uiMode` `UI_MODE_TYPE_MASK` is set to `UI_MODE_TYPE_WATCH`.

The Menu function is deprecated in favor of action bar since Android 4.0. Therefore the new device implementations shipping with Android 6.0 and later MUST NOT implement a dedicated physical button for the Menu function. Older device implementations SHOULD NOT implement a dedicated physical button for the Menu function, but if the physical Menu button is implemented and the device is running applications with `targetSdkVersion > 10`, the device implementation:

- MUST display the action overflow button on the action bar when it is visible and the resulting action overflow menu popup is not empty. For a device implementation launched before Android 4.4 but upgrading to Android 6.0, this is RECOMMENDED.
- MUST NOT modify the position of the action overflow popup displayed by selecting the overflow button in the action bar.
- MAY render the action overflow popup at a modified position on the screen when it is displayed by selecting the physical menu button.

For backwards compatibility, device implementations 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 presented unless hidden together with other navigation functions.

Android device implementations with the support of the Assist action [[Resources, 30](#)] **MUST** make this accessible with a single action (e.g. tap, double-click, or gesture) when other navigation keys are visible, and are **STRONGLY RECOMMENDED** to use the long-press on the Home button or software key as the single action.

Device implementations **MAY** use a distinct portion of the screen to display the navigation keys, but if so, **MUST** meet these requirements:

- Device implementation 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.
- Device implementations **MUST** make available a portion of the display to applications that meets the requirements defined in [section 7.1.1](#).
- Device implementations **MUST** display the navigation keys when applications do not specify a system UI mode, or specify `SYSTEM_UI_FLAG_VISIBLE`.
- Device implementations **MUST** present the navigation keys in an unobtrusive “low profile” (eg. dimmed) mode when applications specify `SYSTEM_UI_FLAG_LOW_PROFILE`.
- Device implementations **MUST** hide the navigation keys when applications specify `SYSTEM_UI_FLAG_HIDE_NAVIGATION`.

7.2.4. Touchscreen Input

 Android Handhelds and Watch Devices **MUST** support touchscreen input.

Device implementations **SHOULD** have a pointer input system of some kind (either mouse-like or touch). However, if a device implementation does not support a pointer input system, it **MUST NOT** report the `android.hardware.touchscreen` or `android.hardware.faketouch` feature constant. Device implementations that do include a pointer input system:

- **SHOULD** support fully independently tracked pointers, if the device input system supports multiple pointers.
- **MUST** report the value of `android.content.res.Configuration.touchscreen` [[Resources, 85](#)] corresponding to the type of the specific touchscreen on the device.

Android includes support for a variety of touchscreens, touch pads, and fake touch input devices. Touchscreen based device implementations are associated with a display [[Resources, 86](#)] 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. In contrast, a 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. Device implementations that declare the fake touch feature **MUST** meet the fake touch requirements in [section 7.2.5](#).

Device implementations **MUST** report the correct feature corresponding to the type of input used. Device implementations that include a touchscreen (single-touch or better) **MUST** report the platform feature constant `android.hardware.touchscreen`. Device implementations that report the platform feature constant `android.hardware.touchscreen` **MUST** also report the platform feature constant `android.hardware.faketouch`. Device implementations that do not include a touchscreen (and rely on a

pointer device only) MUST NOT report any touchscreen feature, and MUST report only android.hardware.faketouch if they meet the fake touch requirements in [section 7.2.5](#).

7.2.5. Fake Touch Input

Device implementations that declare support for android.hardware.faketouch:

- MUST report the absolute X and Y screen positions of the pointer location and display a visual pointer on the screen [[Resources, 87](#)].
- 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 [[Resources, 87](#)].
- MUST support pointer down and up on an object on the screen, which allows users to emulate tap on an object on the screen.
- 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 [[Resources, 87](#)].
- 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.
- 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.

Devices that declare support for android.hardware.faketouch.multitouch.distinct MUST meet the requirements for faketouch above, and MUST also support distinct tracking of two or more independent pointer inputs.

7.2.6. Game Controller Support

Android Television device implementations MUST support button mappings for game controllers as listed below. The upstream Android implementation includes implementation for game controllers that satisfies this requirement.

7.2.6.1. Button Mappings

Android Television device implementations MUST support the following key mappings:

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 [\[Resources, 88\]](#)

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 [\[Resources, 87\]](#)

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 [\[Resources, 87\]](#)

7.2.7. Remote Control

Android Television device implementations SHOULD provide a remote control to allow users to access the TV interface. The remote control MAY be a physical remote or can be a software-based remote that is accessible from a mobile phone or tablet. The remote control MUST meet the requirements defined below.

- **Search affordance.** Device implementations MUST fire KEYCODE_SEARCH when the user invokes voice search either on the physical or software-based remote.
- **Navigation.** All Android Television remotes MUST include Back, Home, and Select buttons and support for D-pad events [\[Resources, 88\]](#).

7.3. Sensors

Android includes APIs for accessing a variety of sensor types. Devices implementations generally MAY omit these sensors, as provided for in the following subsections. If a device includes 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 [\[Resources, 89\]](#). For example, device implementations:

- MUST accurately report the presence or absence of sensors per the android.content.pm.PackageManager class [\[Resources, 70\]](#).
- MUST return an accurate list of supported sensors via the SensorManager.getSensorList() and similar methods.
- 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.).

- 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 [[Resources, 90](#)].
- 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 requirement 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 [[Resources, 91](#)].
- MUST report sensor data with a maximum latency of $100 \text{ milliseconds} + 2 * \text{sample_time}$ for the case of a sensor streamed with a minimum required latency of $5 \text{ ms} + 2 * \text{sample_time}$ when the application processor is active. This delay does not include any filtering delays.
- MUST report the first sensor sample within $400 \text{ milliseconds} + 2 * \text{sample_time}$ of the sensor being activated. It is acceptable for this sample to have an accuracy of 0.

The list above is not comprehensive; the documented behavior of the Android SDK and the Android Open Source Documentations on Sensors [[Resources, 89](#)] 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 [[Resources, 92](#)]. If a device implementation includes a composite sensor it MUST implement the sensor as described in the Android Open Source documentation on composite sensors [[Resources, 92](#)].

Some Android sensors support a “continuous” trigger mode, which returns data continuously [[Resources, 93](#)]. 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.

Note that the device implementations 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.

Finally, when several sensors are activated, the power consumption SHOULD NOT exceed the sum of the individual sensor’s reported power consumption.

7.3.1. Accelerometer

Device implementations SHOULD include a 3-axis accelerometer. Android Handheld devices and Android Watch devices are **STRONGLY RECOMMENDED** to include this sensor. If a device implementation does include a 3-axis accelerometer, it:

- MUST implement and report TYPE_ACCELEROMETER sensor [[Resources, 94](#)].
- MUST be able to report events up to a frequency of at least 50 Hz for Android Watch devices as such devices have a stricter power constraint and 100 Hz for all other device types.
- SHOULD report events up to at least 200 Hz.
- MUST comply with the Android sensor coordinate system as detailed in the Android APIs [[Resources, 90](#)].
- MUST be capable of measuring from freefall up to four times the gravity (4g) or more on any axis.
- MUST have a resolution of at least 12-bits and 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.
- MUST have a standard deviation no greater than 0.05 m/s², 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.
- SHOULD implement the TYPE_SIGNIFICANT_MOTION, TYPE_TILT_DETECTOR, TYPE_STEP_DETECTOR, TYPE_STEP_COUNTER composite sensors as described in the Android SDK document. Existing and new Android devices are **STRONGLY RECOMMENDED** to implement the TYPE_SIGNIFICANT_MOTION composite sensor. If any of these sensors are implemented, the sum of their power consumption MUST always be less than 4 mW and SHOULD each be below 2 mW and 0.5 mW for when the device is in a dynamic or static condition.
- If a gyroscope sensor is included, MUST implement the TYPE_GRAVITY and TYPE_LINEAR_ACCELERATION composite sensors and SHOULD implement the TYPE_GAME_ROTATION_VECTOR composite sensor. Existing and new Android devices are STRONGLY RECOMMENDED to implement the TYPE_GAME_ROTATION_VECTOR sensor.
- MUST implement a TYPE_ROTATION_VECTOR composite sensor, if a gyroscope sensor and a magnetometer sensor is also included.

7.3.2. Magnetometer

Device implementations SHOULD include a 3-axis magnetometer (compass). If a device does include a 3-axis magnetometer, it:

- MUST implement the TYPE_MAGNETIC_FIELD sensor and SHOULD also implement TYPE_MAGNETIC_FIELD_UNCALIBRATED sensor. Existing and new Android devices are STRONGLY RECOMMENDED to implement the TYPE_MAGNETIC_FIELD_UNCALIBRATED sensor.
- 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.
- MUST comply with the Android sensor coordinate system as detailed in the Android APIs [[Resources, 90](#)].
- MUST be capable of measuring between -900 μ T and +900 μ T on each axis before saturating.
- MUST have a hard iron offset value less than 700 μ T and SHOULD have a value below 200 μ T, by placing the magnetometer far from dynamic (current-induced) and static (magnet-induced) magnetic fields.
- MUST have a resolution equal or denser than 0.6 μ T and SHOULD have a resolution equal or denser than 0.2 μ T.
- SHOULD be temperature compensated.
- MUST support online calibration and compensation of the hard iron bias, and preserve the compensation parameters between device reboots.
- MUST have the soft iron compensation applied—the calibration can be done either while in use or during the production of the device.
- SHOULD 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 0.5 μ T.
- MUST implement a TYPE_ROTATION_VECTOR composite sensor, if an accelerometer sensor and a gyroscope sensor is also included.
- MAY implement the TYPE_GEOMAGNETIC_ROTATION_VECTOR sensor if an accelerometer sensor is also implemented. However if implemented, it MUST consume less than 10 mW and 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 receiver. If a device implementation does include a GPS receiver, it SHOULD include some form of “assisted GPS” technique to minimize GPS lock-on time.

7.3.4. Gyroscope

Device implementations SHOULD include a gyroscope (angular change sensor). Devices SHOULD NOT include a gyroscope sensor unless a 3-axis accelerometer is also included. If a device implementation includes a gyroscope, it:

- MUST implement the TYPE_GYROSCOPE sensor and SHOULD also implement TYPE_GYROSCOPE_UNCALIBRATED sensor. Existing and new Android devices are STRONGLY RECOMMENDED to implement the SENSOR_TYPE_GYROSCOPE_UNCALIBRATED sensor.
- MUST be capable of measuring orientation changes up to 1,000 degrees per second.
- MUST be able to report events up to a frequency of at least 50 Hz for Android Watch devices as such devices have a stricter power constraint and 100 Hz for all other device types.
- SHOULD report events up to at least 200 Hz.
- MUST have a resolution of 12-bits or more and SHOULD have a resolution of 16-bits or more.
- MUST be temperature compensated.
- MUST be calibrated and compensated while in use, and preserve the compensation parameters between device reboots.
- MUST have a variance no greater than $1e-7 \text{ rad}^2 / \text{s}^2$ 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 $1e-7 \text{ rad}^2/\text{s}^2$.
- MUST implement a TYPE_ROTATION_VECTOR composite sensor, if an accelerometer sensor and a magnetometer sensor is also included.
- If an accelerometer sensor is included, MUST implement the TYPE_GRAVITY and TYPE_LINEAR_ACCELERATION composite sensors and SHOULD implement the TYPE_GAME_ROTATION_VECTOR composite sensor. Existing and new Android devices are STRONGLY RECOMMENDED to implement the TYPE_GAME_ROTATION_VECTOR sensor.

7.3.5. Barometer

Device implementations SHOULD include a barometer (ambient air pressure sensor). If a device implementation includes a barometer, it:

- MUST implement and report TYPE_PRESSURE sensor.
- MUST be able to deliver events at 5 Hz or greater.
- MUST have adequate precision to enable estimating altitude.
- MUST be temperature compensated.

7.3.6. Thermometer

Device implementations MAY include an ambient thermometer (temperature sensor). If present, it MUST be defined as `SENSOR_TYPE_AMBIENT_TEMPERATURE` and it MUST measure the ambient (room) temperature in degrees Celsius.

Device implementations MAY but SHOULD NOT include a CPU temperature sensor. If present, it MUST be defined as `SENSOR_TYPE_TEMPERATURE`, it MUST measure the temperature of the device CPU, and it 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. Devices that can make a voice call and indicate any value other than `PHONE_TYPE_NONE` in `getPhoneType` SHOULD include a proximity sensor. If a device implementation does include a proximity sensor, it:

- 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 a device implementation includes a proximity sensor with any other orientation, it MUST NOT be accessible through this API.
- MUST have 1-bit of accuracy or more.

7.3.9. High Fidelity Sensors

Device implementations supporting a set of higher quality sensors that can meet all the requirements listed in this section MUST identify the support through the `android.hardware.sensor.hifi_sensors` feature flag.

A device declaring `android.hardware.sensor.hifi_sensors` MUST support all of the following sensor types meeting the quality requirements as below:

- `SENSOR_TYPE_ACCELEROMETER`
 - 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 200 Hz or higher
 - MUST have a measurement noise not above 400uG/√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
- `SENSOR_TYPE_GYROSCOPE`
 - 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 200 Hz or higher
 - MUST have a measurement noise not above 0.014°/s/√Hz
- `SENSOR_TYPE_GYROSCOPE_UNCALIBRATED` with the same quality requirements as `SENSOR_TYPE_GYROSCOPE`
- `SENSOR_TYPE_GEOMAGNETIC_FIELD`

- 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
- **SENSOR_TYPE_MAGNETIC_FIELD_UNCALIBRATED** with the same quality requirements as **SENSOR_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
- **SENSOR_TYPE_PRESSURE**
 - 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/ $\sqrt{\text{Hz}}$
 - 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
- **SENSOR_TYPE_ROTATION_VECTOR**
 - MUST have a batching power consumption not worse than 4 mW
- **SENSOR_TYPE_GAME_ROTATION_VECTOR** MUST implement a non-wake-up form of this sensor with a buffering capability of at least 300 sensor events
- **SENSOR_TYPE_SIGNIFICANT_MOTION**
 - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving
- **SENSOR_TYPE_STEP_DETECTOR**
 - 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
- **SENSOR_TYPE_STEP_COUNTER**
 - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving
- **SENSOR_TILT_DETECTOR**
 - MUST have a power consumption not worse than 0.5 mW when device is static and 1.5 mW when device is moving

Also such a device MUST meet the following sensor subsystem requirements:

- 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.
- The Gyroscope sensor event timestamps MUST be on the same time base as the camera subsystem and within 1 milliseconds of error.
- The latency of delivery of samples to the HAL SHOULD be below 5 milliseconds from the instant the data is available on the physical sensor hardware.
- The power consumption MUST not be 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`

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.

The following sensor types MAY also be supported on a device implementation declaring `android.hardware.sensor.hifi_sensors`, but if these sensor types are present they MUST meet the following minimum buffering capability requirement:

- `SENSOR_TYPE_PROXIMITY`: 100 sensor events

7.3.10. Fingerprint Sensor

Device implementations with a secure lock screen SHOULD include a fingerprint sensor. If a device implementation includes a fingerprint sensor and has a corresponding API for third-party developers, it:

- MUST declare support for the `android.hardware.fingerprint` feature.
- MUST fully implement the corresponding API as described in the Android SDK documentation [[Resources, 95](#)].
- MUST have a false acceptance rate not higher than 0.002%.
- Is STRONGLY RECOMMENDED to have a false rejection rate not higher than 10%, and a latency from when the fingerprint sensor is touched until the screen is unlocked below 1 second, for 1 enrolled finger.
- MUST rate limit attempts for at least 30 seconds after 5 false trials for fingerprint verification.
- 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.
- 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 [[Resources, 96](#)].
- 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) using the TEE as implemented in the Android Open Source project.
- MUST NOT enable 3rd-party applications to distinguish between individual fingerprints.
- MUST honor the `DevicePolicyManager.KEYGUARD_DISABLE_FINGERPRINT` flag.
- MUST, when upgraded from a version earlier than Android 6.0, have the fingerprint data securely migrated to meet the above requirements or removed.
- SHOULD use the Android Fingerprint icon provided in the Android Open Source Project.

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 MUST NOT report the `android.hardware.telephony` feature or any subfeatures, 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. However, if a device implementation does include GSM or CDMA telephony, it MUST implement full support for the API for that technology. Device implementations that do not include telephony hardware MUST implement the full APIs as no-ops.

7.4.2. IEEE 802.11 (Wi-Fi)

Android Television device implementations MUST include Wi-Fi support.

Android Television device implementations MUST include support for one or more forms of 802.11 (b/g/a/n, etc.) and other types of Android device implementation SHOULD include support for one or more forms of 802.11. If a device implementation does include support for 802.11 and exposes the functionality to a third-party application, it MUST implement the corresponding Android API and:

- MUST report the hardware feature flag `android.hardware.wifi`.
- MUST implement the multicast API as described in the SDK documentation [[Resources, 97](#)].
- 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.

7.4.2.1. Wi-Fi Direct

Device implementations SHOULD include support for Wi-Fi Direct (Wi-Fi peer-to-peer). If a device implementation does include support for Wi-Fi Direct, it MUST implement the corresponding Android API as described in the SDK documentation [[Resources, 98](#)]. If a device implementation includes support for Wi-Fi Direct, then it:

- MUST report the hardware feature `android.hardware.wifi.direct`.
- MUST support regular Wi-Fi operation.
- SHOULD support concurrent Wi-Fi and Wi-Fi Direct operation.

7.4.2.2. Wi-Fi Tunneled Direct Link Setup

Android Television device implementations MUST include support for Wi-Fi Tunneled Direct Link Setup (TDLS).

Android Television device implementations MUST include support for Wi-Fi Tunneled Direct Link Setup (TDLS) and other types of Android device implementations SHOULD include support for Wi-Fi TDLS as described in the Android SDK Documentation [[Resources, 99](#)]. If a device implementation does include support for TDLS and TDLS is enabled by the `WiFiManager` API, the device:

- 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.

7.4.3. Bluetooth

Android Watch and Automotive implementations MUST support Bluetooth. Android Television

■ implementations MUST support Bluetooth and Bluetooth LE.

Android includes support for Bluetooth and Bluetooth Low Energy [[Resources, 100](#)]. Device implementations that include support for Bluetooth and Bluetooth Low Energy MUST declare the relevant platform features (`android.hardware.bluetooth` and `android.hardware.bluetooth_le` respectively) and implement the platform APIs. Device implementations SHOULD implement relevant Bluetooth profiles such as A2DP, AVCP, OBEX, etc. as appropriate for the device. Android Television device implementations MUST support Bluetooth and Bluetooth LE.

Device implementations including support for Bluetooth Low Energy:

- MUST declare the hardware feature `android.hardware.bluetooth_le`.
- MUST enable the GATT (generic attribute profile) based Bluetooth APIs as described in the SDK documentation and [[Resources, 100](#)].
- MUST implement a Resolvable Private Address (RPA) timeout no longer than 15 minutes, and rotate the address at timeout to protect user privacy.
- SHOULD support offloading of the filtering logic to the bluetooth chipset when implementing the `ScanFilter` API [[Resources, 101](#)], and MUST report the correct value of where the filtering logic is implemented whenever queried via the `android.bluetooth.BluetoothAdapter.isOffloadedFilteringSupported()` method.
- SHOULD support offloading of the batched scanning to the bluetooth chipset, but if not supported, MUST report 'false' whenever queried via the `android.bluetooth.BluetoothAdapter.isOffloadedScanBatchingSupported()` method.
- SHOULD support multi advertisement with at least 4 slots, but if not supported, MUST report 'false' whenever queried via the `android.bluetooth.BluetoothAdapter.isMultipleAdvertisementSupported()` method.

7.4.4. Near-Field Communications

Device implementations SHOULD include a transceiver and related hardware for Near-Field Communications (NFC). If a device implementation does include NFC hardware and plans to make it available to third-party apps, then it:

- MUST report the `android.hardware.nfc` feature from the `android.content.pm.PackageManager.hasSystemFeature()` method [[Resources, 70](#)].
- MUST be capable of reading and writing NDEF messages via the following NFC standards:
 - 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 (defined by the NFC Forum)
 - 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 below 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.
 - NfcV (ISO 15693)

- SHOULD be capable of reading the barcode and URL (if encoded) of Thinfilm NFC Barcode [[Resources, 102](#)] products.
- 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 [[Resources, 103](#)]
 - SNEP 1.0 (defined by the NFC Forum)
- MUST include support for Android Beam [[Resources, 104](#)]:
 - 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.
 - MUST honor the `android.settings.NFCSHARING_SETTINGS` intent to show NFC sharing settings [[Resources, 105](#)].
 - MUST implement the NPP server. Messages received by the NPP server MUST be processed the same way as the SNEP default server.
 - 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.
 - 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.
 - SHOULD enable Android Beam by default and MUST be able to send and receive using Android Beam, even when another proprietary NFC P2p mode is turned on.
 - MUST support NFC Connection handover to Bluetooth when the device supports Bluetooth Object Push Profile. Device implementations MUST support connection handover to Bluetooth when using `android.nfc.NfcAdapter.setBeamPushUris`, by implementing the "Connection Handover version 1.2" [[Resources, 106](#)] and "Bluetooth Secure Simple Pairing Using NFC version 1.0" [[Resources, 107](#)] 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.
- 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.

(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 a device implementation does include an NFC controller chipset capable of HCE and Application ID (AID) routing, then it:

- MUST report the `android.hardware.nfc.hce` feature constant.
- MUST support NFC HCE APIs as defined in the Android SDK [[Resources, 108](#)].

Additionally, device implementations MAY include reader/writer support for the following MIFARE technologies.

- MIFARE Classic
- MIFARE Ultralight
- NDEF on MIFARE Classic

Note that Android includes APIs for these MIFARE types. If a device implementation supports MIFARE in the reader/writer role, it:

- MUST implement the corresponding Android APIs as documented by the Android SDK.
- MUST report the feature `com.nxp.mifare` from the `android.content.pm.PackageManager.hasSystemFeature()` method [[Resources, 70](#)]. 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.
- MUST NOT implement the corresponding Android APIs nor report the `com.nxp.mifare` feature unless it also implements general NFC support as described in this section.

If a device implementation does not include NFC hardware, it MUST NOT declare the `android.hardware.nfc` feature from the `android.content.pm.PackageManager.hasSystemFeature()` method [[Resources, 70](#)], and MUST implement the Android NFC API as a no-op.

As the classes `android.nfc.NdefMessage` and `android.nfc.NdefRecord` represent a protocol-independent data representation format, device implementations MUST implement these APIs even if they do not include support for NFC or declare the `android.hardware.nfc` feature.

7.4.5. Minimum Network Capability

Device implementations 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.

Device implementations where a physical networking standard (such as Ethernet) is the primary data connection SHOULD also include support for at least one common wireless data standard, such as 802.11 (Wi-Fi).

Devices MAY implement more than one form of data connectivity.

Devices 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. The required level of IPv6 support depends on the network type, as follows:

- Devices that support Wi-Fi networks MUST support dual-stack and IPv6-only operation on Wi-Fi.
- Devices that support Ethernet networks MUST support dual-stack operation on Ethernet.
- Devices that support cellular data SHOULD support IPv6 operation (IPv6-only and

possibly dual-stack) on cellular data.

- When a device is simultaneously connected to more than one network (e.g., Wi-Fi and cellular data), it **MUST** simultaneously meet these requirements on each network to which it is connected.

IPv6 **MUST** be enabled by default.

In order to ensure that IPv6 communication is as reliable as IPv4, unicast IPv6 packets sent to the device **MUST NOT** be dropped, even when the screen is not in an active state. Redundant multicast IPv6 packets, such as repeated identical Router Advertisements, **MAY** be rate-limited in hardware or firmware if doing so is necessary to save power. In such cases, 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.

IPv6 connectivity **MUST** be maintained in doze mode.

7.4.6. Sync Settings

Device implementations **MUST** have the master auto-sync setting on by default so that the method `getMasterSyncAutomatically()` returns “true” [[Resources, 109](#)].

7.5. Cameras

Device implementations **SHOULD** include a rear-facing camera and **MAY** include a front-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. 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.

If a device implementation includes at least one camera, it **SHOULD** be possible for an application to simultaneously allocate 3 bitmaps equal to the size of the images produced by the largest-resolution camera sensor on the device.

7.5.1. Rear-Facing Camera

Device implementations **SHOULD** include a rear-facing camera. If a device implementation includes at least one rear-facing camera, it:

- **MUST** report the feature flag `android.hardware.camera` and `android.hardware.camera.any`.
- **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, 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

Device implementations **MAY** include a front-facing camera. If a device implementation includes at least one front-facing camera, it:

- **MUST** report the feature flag `android.hardware.camera.any` and

android.hardware.camera.front.

- MUST have a resolution of at least VGA (640x480 pixels).
- MUST NOT use a front-facing camera as the default for the Camera API. The camera API in Android has specific support for front-facing cameras and device implementations 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.
- MAY include features (such as auto-focus, flash, etc.) available to rear-facing cameras as described in [section 7.5.1](#).
- MUST horizontally reflect (i.e. mirror) the stream displayed by an app in a CameraPreview, as follows:
 - If the device implementation is capable of being rotated by user (such as automatically via an accelerometer or manually via user input), the camera preview MUST be mirrored horizontally relative to the device's current orientation.
 - If the current application has explicitly requested that the Camera display be rotated via a call to the `android.hardware.Camera.setDisplayOrientation()` [[Resources, 110](#)] method, the camera preview MUST be mirrored horizontally relative to the orientation specified by the application.
 - Otherwise, the preview MUST be mirrored along the device's default horizontal axis.
- MUST mirror the image displayed by the postview in the same manner as the camera preview image stream. If the device implementation does not support postview, this requirement obviously does not apply.
- MUST NOT mirror the final captured still image or video streams returned to application callbacks or committed to media storage.

7.5.3. External Camera

Device implementations with USB host mode MAY include support for an external camera that connects to the USB port. If a device includes support for an external camera, it:

- MUST declare the platform feature `android.hardware.camera.external` and `android.hardware.camera.any`.
- MUST support USB Video Class (UVC 1.0 or higher).
- MAY support multiple cameras.

Video compression (such as MJPEG) support is RECOMMENDED to enable transfer of high-quality unencoded streams (i.e. raw or independently compressed picture streams). Camera-based video encoding MAY be supported. If so, 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:

- If an application has never called `android.hardware.Camera.Parameters.setPreviewFormat(int)`, then the device **MUST** use `android.hardware.PixelFormat.YCbCr_420_SP` for preview data provided to application callbacks.
- If an application registers an `android.hardware.Camera.PreviewCallback` instance and the system calls the `onPreviewFrame()` method when the preview format is `YCbCr_420_SP`, the data in the `byte[]` passed into `onPreviewFrame()` must further be in the NV21 encoding format. That is, NV21 **MUST** be the default.
- For `android.hardware.Camera`, device implementations **MUST** support the YV12 format (as denoted by the `android.graphics.ImageFormat.YV12` constant) for camera previews for both front- and rear-facing cameras. (The hardware video encoder and camera may use any native pixel format, but the device implementation **MUST** support conversion to YV12.)
- For `android.hardware.camera2`, device implementations must support the `android.hardware.ImageFormat.YUV_420_888` and `android.hardware.ImageFormat.JPEG` formats as outputs through the `android.media.ImageReader` API.

Device implementations **MUST** still implement the full Camera API included in the Android SDK documentation [[Resources, 111](#)], 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.

Device implementations **MUST** recognize and honor each parameter name defined as a constant on the `android.hardware.Camera.Parameters` class, if the underlying hardware supports the feature. If the device hardware does not support a feature, the API must behave as documented. 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` [[Resources, 112](#)].

Because not all device implementations can fully support all the features of the `android.hardware.camera2` API, device implementations **MUST** report the proper level of support with the `android.info.supportedHardwareLevel` property as described in the Android SDK [[Resources, 113](#)] and report the appropriate framework feature flags [[Resources, 114](#)].

Device implementations **MUST** also declare its Individual camera capabilities of `android.hardware.camera2` via the `android.request.availableCapabilities` property and declare the appropriate feature flags [[Resources, 114](#)]; a device must define the feature flag if any of its attached camera devices supports the feature.

Device implementations **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.

Device implementations **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

Both front- and rear-facing cameras, if present, **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

Android Television devices **MUST** have at least 5GB of non-volatile storage available for application private data.

The memory available to the kernel and userspace on device implementations **MUST** be at least equal or larger than the minimum values specified by the following table. (See [section 7.1.1](#) for screen size and density definitions.)

Density and screen size	32-bit device	64-bit device
Android Watch devices (due to smaller screens)	416MB	Not applicable
<ul style="list-style-type: none">• 280dpi or lower on small/normal screens• mdpi or lower on large screens• ldpi or lower on extra large screens	424MB	704MB
<ul style="list-style-type: none">• xhdpi or higher on small/normal screens• hdpi or higher on large screens• mdpi or higher on extra large screens	512MB	832MB
<ul style="list-style-type: none">• 400dpi or higher on small/normal screens• xhdpi or higher on large screens• tvdpi or higher on extra large screens	896MB	1280MB
<ul style="list-style-type: none">• 560dpi or higher on small/normal screens• 400dpi or higher on large screens• xhdpi or higher on extra large screens	1344MB	1824MB

The minimum memory values **MUST** be in addition to any memory space already dedicated to hardware components such as radio, video, and so on that is not under the kernel's control.

Device implementations with less than 512MB of memory available to the kernel and userspace, unless an Android Watch, **MUST** return the value "true" for `ActivityManager.isLowRamDevice()`.

Android Television devices **MUST** have at least 5GB and other device implementations **MUST** have at least 1.5GB of non-volatile storage available for application private data. That is, the `/data` partition **MUST** be at least 5GB for Android Television devices and at least 1.5GB for other device implementations. Device implementations that run Android are **STRONGLY RECOMMENDED** to have at least 3GB of non-volatile storage for application private data so they will be able to upgrade to the future platform releases.

The Android APIs include a Download Manager that applications **MAY** use to download data files [[Resources, 115](#)]. The device implementation of the Download Manager **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 **MUST** offer shared storage for applications also often referred as "shared external storage".

Device implementations **MUST** be configured with shared storage mounted by default, “out of the box”. If the shared storage is not mounted on the Linux path /sdcard, then the device **MUST** include a Linux symbolic link from /sdcard to the actual mount point.

Device implementations **MAY** have hardware for user-accessible removable storage, such as a Secure Digital (SD) card slot. If this slot is used to satisfy the shared storage requirement, the device implementation:

- **MUST** implement a toast or pop-up user interface warning the user when there is no SD card.
- **MUST** include a FAT-formatted SD card 1GB in size or larger **OR** show on the box and other material available at time of purchase that the SD card has to be separately purchased.
- **MUST** mount the SD card by default.

Alternatively, device implementations **MAY** allocate internal (non-removable) storage as shared storage for apps as included in the upstream Android Open Source Project; device implementations **SHOULD** use this configuration and software implementation. If a device implementation uses internal (non-removable) storage to satisfy the shared storage requirement, that storage **MUST** be 1GB in size or larger and mounted on /sdcard (or /sdcard **MUST** be a symbolic link to the physical location if it is mounted elsewhere).

Device implementations **MUST** enforce as documented the android.permission.WRITE_EXTERNAL_STORAGE permission on this shared storage. Shared storage **MUST** otherwise be writable by any application that obtains that permission.

Device implementations that include multiple shared storage paths (such as both an SD card slot and shared internal storage) **MUST NOT** allow Android applications to write to the secondary external storage, except for their package-specific directories on the secondary external storage, but **SHOULD** expose content from both storage paths transparently through Android’s media scanner service and android.provider.MediaStore.

Regardless of the form of shared storage used, if the device implementation has a USB port with USB peripheral mode support, it **MUST** provide some mechanism to access the contents of shared storage from a host computer. Device implementations **MAY** use USB mass storage, but **SHOULD** use Media Transfer Protocol to satisfy this requirement. If the device implementation supports Media Transfer Protocol, it:

- **SHOULD** be compatible with the reference Android MTP host, Android File Transfer [[Resources, 116](#)].
- **SHOULD** report a USB device class of 0x00.
- **SHOULD** report a USB interface name of 'MTP'.

7.6.3. Adoptable Storage

Device implementations are **STRONGLY RECOMMENDED** to implement adoptable storage if the removable storage device port is in a long-term stable location, such as within the battery compartment or other protective cover [[Resources, 117](#)].

Device implementations such as a television, **MAY** enable adoption through USB ports as the device is expected to be static and not mobile. But for other device implementations that are mobile in nature, it is **STRONGLY RECOMMENDED** to implement the adoptable storage in a long-term stable location, since accidentally disconnecting them can cause data loss/corruption.

7.7. USB

Device implementations **SHOULD** support USB peripheral mode and **SHOULD** support USB host mode.

If a device implementation includes a USB port supporting peripheral mode:

- The port **MUST** be connectable to a USB host that has a standard type-A or type -C USB port.
- The port **SHOULD** use micro-A, 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.
- The port **SHOULD** be centered in the middle of an edge. Device implementations **SHOULD** either locate the port 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.
- It **MUST** allow a USB host connected with the Android device to access the contents of the shared storage volume using either USB mass storage or Media Transfer Protocol.
- It **SHOULD** implement the Android Open Accessory (AOA) API and specification as documented in the Android SDK documentation, and if it is an Android Handheld device it **MUST** implement the AOA API. Device implementations implementing the AOA specification:
 - **MUST** declare support for the hardware feature `android.hardware.usb.accessory` [[Resources, 118](#)].
 - **MUST** implement the USB audio class as documented in the Android SDK documentation [[Resources, 119](#)].
 - And also the USB mass storage class, **MUST** include the string "android" at the end of the interface description `iInterface` string of the USB mass storage
- It **SHOULD** implement support to draw 1.5 A current during HS chirp and traffic as specified in the USB battery charging specification [[Resources, 120](#)]. Existing and new Android devices are **STRONGLY RECOMMENDED to meet these requirements** so they will be able to upgrade to the future platform releases.
- The value of `iSerialNumber` in USB standard device descriptor **MUST** be equal to the value of `android.os.Build.SERIAL`.

If a device implementation includes a USB port supporting host mode, it:

- **SHOULD** use a type-C USB port, if the device implementation supports USB 3.1.
- **MAY** use a non-standard port form factor, but if so **MUST** ship with a cable or cables adapting the port to a standard type-A or type-C USB port.
- **MAY** use a micro-AB USB port, but if so **SHOULD** ship with a cable or cables adapting the port to a standard type-A or type-C USB port.
- is **STRONGLY RECOMMENDED** to implement the USB audio class as documented in the Android SDK documentation [[Resources, 119](#)].
- **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` [[Resources, 121](#)].
- **SHOULD** support the Charging Downstream Port output current range of 1.5 A ~ 5 A as specified in the USB Battery Charging Specifications [[Resources, 120](#)].

7.8. Audio

7.8.1. Microphone

- Android Handheld, Watch, and Automotive implementations **MUST** include a microphone.

Device implementations MAY omit a microphone. However, if a device implementation omits a microphone, it MUST NOT report the `android.hardware.microphone` feature constant, and MUST implement the audio recording API at least as no-ops, per [section 7](#). Conversely, device implementations that do possess a microphone:

- MUST report the `android.hardware.microphone` feature constant
- MUST meet the audio recording requirements in [section 5.4](#)
- MUST meet the audio latency requirements in [section 5.6](#)
- STRONGLY RECOMMENDED to support near-ultrasound recording as described in [section 7.8.3](#)

7.8.2. Audio Output

 Android Watch devices MAY include an audio output.

Device implementations including a speaker or with an audio/multimedia output port for an audio output peripheral as a headset or an external speaker:

- MUST report the `android.hardware.audio.output` feature constant.
- MUST meet the audio playback requirements in [section 5.5](#).
- MUST meet the audio latency requirements in [section 5.6](#).
- STRONGLY RECOMMENDED to support near-ultrasound playback as described in [section 7.8.3](#)

Conversely, if a device implementation does not include a speaker or audio output port, it MUST NOT report the `android.hardware.audio.output` feature, and MUST implement the Audio Output related APIs as no-ops at least.

Android Watch device implementation MAY but SHOULD NOT have audio output, but other types of Android device implementations MUST have an audio output and declare `android.hardware.audio.output`.

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 [[Resources, 122](#)], 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 a device implementation has a 4 conductor 3.5mm audio jack, it:

- MUST support audio playback to stereo headphones and stereo headsets with a microphone, and SHOULD support audio recording from stereo headsets with a microphone.
- MUST support TRRS audio plugs with the CTIA pin-out order, and SHOULD support audio plugs with the OMTP pin-out order.
- MUST support the detection of microphone on the plugged in audio accessory, if the device implementation supports a microphone, and broadcast the `android.intent.action.HEADSET_PLUG` with the extra value `microphone` set as 1.
- SHOULD 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`
- SHOULD support the detection and mapping 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
- MUST trigger ACTION_HEADSET_PLUG upon a plug insert, but only after all contacts on plug are touching their relevant segments on the jack.
- MUST be capable of driving at least 150mV ± 10% of output voltage on a 32 Ohm speaker impedance.
- MUST have a microphone bias voltage between 1.8V ~ 2.9V.

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", then
 - 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.
 - The signal to noise ratio of the microphone MUST be no lower than 80 dB.
- If [PROPERTY_SUPPORT_SPEAKER_NEAR_ULTRASOUND](#) is "true", then the speaker's mean response in 18.5 kHz - 20 kHz MUST be no lower than 40 dB below the response at 2 kHz.

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. Android Watch devices SHOULD and other type of device implementations MUST meet the following criteria:

8.1. User Experience Consistency

Device implementations MUST provide a smooth user interface by ensuring a consistent frame rate and response times for applications and games. Device implementations MUST meet the following requirements:

- **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.
- **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.
- **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.

8.2. File I/O Access Performance

Device implementations MUST ensure internal storage file access performance consistency for read and write operations.

- **Sequential write.** Device implementations MUST ensure a sequential write performance of at least 5MB/s for a 256MB file using 10MB write buffer.
- **Random write.** Device implementations MUST ensure a random write performance of at least 0.5MB/s for a 256MB file using 4KB write buffer.
- **Sequential read.** Device implementations MUST ensure a sequential read performance of at least 15MB/s for a 256MB file using 10MB write buffer.

- **Random read.** Device implementations MUST ensure a random read performance of at least 3.5MB/s for a 256MB file using 4KB write buffer.

8.3. Power-Saving Modes

All apps exempted from App Standby and/or Doze mode MUST be made visible to the end user. Further, the triggering, maintenance, wakeup algorithms and the use of Global system settings of these power-saving modes MUST not deviate from the Android Open Source Project.

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. Therefore, device implementations:

- MUST be able to track hardware component power usage and attribute that power usage to specific applications. Specifically, implementations:
 - 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 [[Resources, 123](#)].
 - MUST report all power consumption values in milliampere hours (mAh)
 - SHOULD be attributed to the hardware component itself if unable to attribute hardware component power usage to an application.
 - 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.
- MUST make this power usage available via the `adb shell dumpsys batterystats` shell command to the app developer [[Resources, 124](#)].
- MUST honor the `android.intent.action.POWER_USAGE_SUMMARY` intent and display a settings menu that shows this power usage [[Resources, 125](#)].

9. Security Model Compatibility

Device implementations MUST implement a security model consistent with the Android platform security model as defined in Security and Permissions reference document in the APIs [[Resources, 126](#)] in the Android developer documentation. Device implementations 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 MUST support the Android permissions model as defined in the Android developer documentation [[Resources, 126](#)]. Specifically, implementations MUST enforce each permission defined as described in the SDK documentation; no permissions may be omitted, altered, or ignored. Implementations MAY add additional permissions, provided the new permission ID strings are not in the `android.*` namespace.

Permissions with a protection level of `dangerous` are runtime permissions. Applications with `targetSdkVersion > 22` request them at runtime. Device implementations:

- 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.

- MUST have one and only one implementation of both user interfaces.
- 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

9.2. UID and Process Isolation

Device implementations MUST support the Android application sandbox model, in which each application runs as a unique Unixstyle UID and in a separate process. Device implementations 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 [[Resources, 126](#)].

9.3. Filesystem Permissions

Device implementations MUST support the Android file access permissions model as defined in the Security and Permissions reference [[Resources, 126](#)].

9.4. Alternate Execution Environments

Device implementations MAY include runtime environments that execute applications using some other software or technology than the Dalvik Executable Format or native code. However, such alternate execution environments MUST NOT compromise the Android security model or the security of installed Android applications, as described in this section.

Alternate runtimes MUST themselves be Android applications, and abide by the standard Android security model, as described elsewhere in [section 9](#).

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.

Alternate runtimes MUST NOT permit applications to make use of features protected by Android permissions restricted to system applications.

Alternate runtimes MUST abide by the Android sandbox model. Specifically, alternate runtimes:

- SHOULD install apps via the PackageManager into separate Android sandboxes (Linux user IDs, etc.).
- MAY provide a single Android sandbox shared by all applications using the alternate runtime.
- 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.
- MUST NOT launch with, grant, or be granted access to the sandboxes corresponding to other Android applications.
- MUST NOT be launched with, be granted, or grant to other applications any privileges of the superuser (root), or of any other user ID.

The .apk files of alternate runtimes MAY be included in the system image of a device implementation, but MUST be signed with a key distinct from the key used to sign other applications included with the device implementation.

When installing applications, alternate runtimes MUST obtain user consent for the Android

permissions used by the application. If 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. If 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.

9.5. Multi-User Support

■ This feature is optional for all device types.

Android includes support for multiple users and provides support for full user isolation [[Resources, 127](#)]. Device implementations MAY enable multiple users, but when enabled MUST meet the following requirements related to multi-user support [[Resources, 128](#)]:

- Device implementations that do not declare the `android.hardware.telephony` feature flag 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.
- Conversely device implementations that declare the `android.hardware.telephony` feature flag 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.
- Device implementations 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 [[Resources, 126](#)].
- Each user instance on an Android device MUST have separate and isolated external storage directories. Device implementations MAY store multiple users' data on the same volume or filesystem. However, the device implementation MUST ensure that applications owned by and running on behalf a given user cannot list, read, or write to data owned by any other user. Note that removable media, such as SD card slots, can allow one user to access another's data by means of a host PC. For this reason, device implementations that use removable media for the external storage APIs MUST encrypt the contents of the SD card if multiuser is enabled using a key stored only on non-removable media accessible only to the system. 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. Accordingly, device implementations MAY but SHOULD NOT enable multi-user if they use removable media [[Resources, 129](#)] for primary external storage.

9.6. Premium SMS Warning

Android includes support for warning users of any outgoing premium SMS message [[Resources, 130](#)]. Premium SMS messages are text messages sent to a service registered with a carrier that may incur a charge to the user. Device implementations that declare support for `android.hardware.telephony` 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 and other security features in the Linux kernel. SELinux or any other security features implemented below the Android framework:

- MUST maintain compatibility with existing applications.
- MUST NOT have a visible user interface when a security violation is detected and successfully blocked, but MAY have a visible user interface when an unblocked security violation occurs resulting in a successful exploit.
- SHOULD NOT be user or developer configurable.

If any API for configuration of policy is exposed to an application that can affect another application (such as a Device Administration API), the API MUST NOT allow configurations that break compatibility.

Devices MUST implement SELinux or, if using a kernel other than Linux, an equivalent mandatory access control system. Devices MUST also meet the following requirements, which are satisfied by the reference implementation in the upstream Android Open Source Project.

Device implementations:

- MUST set SELinux to global enforcing mode.
- MUST configure all domains in enforcing mode. No permissive mode domains are allowed, including domains specific to a device/vendor.
- MUST NOT modify, omit, or replace the neverallow rules present within the external/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.

Device implementations SHOULD retain the default SELinux policy provided in the external/sepolicy folder of the upstream Android Open Source Project and only further add to this policy for their own device-specific configuration. Device implementations MUST be compatible with the upstream Android Open Source Project.

9.8. Privacy

If the device implements functionality in the system that captures the contents displayed on the screen and/or records the audio stream played on the device, it MUST continuously notify the user whenever this functionality is enabled and actively capturing/recording.

If a device implementation has a mechanism that routes network data traffic through a proxy server or VPN gateway by default (for example, preloading a VPN service with `android.permission.CONTROL_VPN` granted), the device implementation MUST ask for the user's consent before enabling that mechanism.

If a device implementation has a USB port with USB peripheral mode support, it 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.

9.9. Full-Disk Encryption

 Optional for Android device implementations without a lock screen.

If the device implementation supports a secure lock screen reporting `"true"` for `KeyguardManager.isDeviceSecure()` [[Resources, 131](#)], and is not a device with restricted memory as reported through the `ActivityManager.isLowRamDevice()` method, then the device MUST support full-disk encryption [[Resources, 132](#)] 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.

For device implementations supporting full-disk encryption and with Advanced Encryption Standard (AES) crypto performance above 50MiB/sec, the full-disk encryption MUST be enabled by default at the time the user has completed the out-of-box setup experience. If a device implementation is

already launched on an earlier Android version with full-disk encryption disabled by default, such a device cannot meet the requirement through a system software update and thus MAY be exempted. Encryption MUST use AES with a key of 128-bits (or greater) and a mode designed for storage (for example, AES-XTS, AES-CBC-ESSIV). The encryption key MUST NOT be written to storage at any time without being encrypted. Other than when in active use, the encryption key SHOULD be AES encrypted with the lockscreen passcode stretched using a slow stretching algorithm (e.g. PBKDF2 or scrypt). If the user has not specified a lockscreen passcode or has disabled use of the passcode for encryption, the system SHOULD use a default passcode to wrap the encryption key. If the device provides a hardware-backed keystore, the password stretching algorithm MUST be cryptographically bound to that keystore. The encryption key MUST NOT be sent off 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. Verified Boot

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

- Declare the platform feature flag `android.software.verified_boot`
- Perform verification on every boot sequence
- Start verification from an immutable hardware key that is the root of trust, and go all the way up to the system partition
- 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
- Use verification algorithms as strong as current recommendations from NIST for hashing algorithms (SHA-256) and public key sizes (RSA-2048)

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

Starting from Android 6.0, device implementations with Advanced Encryption Standard (AES) crypto performance above 50MiB/seconds 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 [[Resources, 133](#)] allows app developers to store cryptographic keys in a container and use them in cryptographic operations through the KeyChain API [[Resources, 134](#)] or the Keystore API [[Resources, 135](#)].

All Android device implementations MUST meet the following requirements:

- SHOULD not limit the number of keys that can be generated, and MUST at least allow more than 8,192 keys to be imported.
- The lock screen authentication MUST rate limit attempts and SHOULD have an exponential backoff algorithm as implemented in the Android Open Source Project.
- When the device implementation supports a secure lock screen and has a secure hardware such as a Secure Element (SE) where a Trusted Execution Environment (TEE) can be implemented, then it:
 - Is **STRONGLY RECOMMENDED** to back up the keystore implementation with the secure hardware. The upstream Android Open Source Project provides the Keymaster Hardware Abstraction Layer (HAL) implementation that can be used to satisfy this requirement.

- MUST perform the lock screen authentication in the secure hardware if the device has a hardware-backed keystore implementation 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) that can be used to satisfy this requirement [[Resources, 136](#)].

Note that while the above TEE-related requirements are stated as STRONGLY RECOMMENDED, the Compatibility Definition for the next API version is planned to change these to REQUIRED. If a device implementation is already launched on an earlier Android version and has not implemented a trusted operating system on the secure hardware, such a device might not be able to meet the requirements through a system software update and thus is STRONGLY RECOMMENDED to implement a TEE.

9.12. Data Deletion

Devices MUST provide users with a mechanism to perform a "Factory Data Reset" that allows logical and physical deletion of all data. This MUST satisfy relevant industry standards for data deletion such as NIST SP800-88. This MUST be used for the implementation of the wipeData() API (part of the Android Device Administration API) described in [section 3.9 Device Administration](#).

Devices MAY provide a fast data wipe that conducts a logical data erase.

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) [[Resources, 137](#)] 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 6.0. 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:

- Android Automotive implementations **SHOULD** support OTA downloads with offline update via reboot.
- All other device implementations **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.

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 [[Resources, 138](#)].

12. Document Changelog

The following table contains a summary of the changes to the Compatibility Definition in this release.

Section	Summary of changes
Various	Replaced instances of the "encouraged" term with "RECOMMENDED"
2. Device Types	Update for Android Automotive implementations
3.2.2. Build Parameters	Additions for the hardware serial number and for the security patch level of a build

3.2.3.2. Intent Resolution	Section renamed from "Intent Overrides" to "Intent Resolution," with new requirements related to authoritative default app linking
3.3.1. Application Binary Interfaces	Additions for Android ABI support; change related to Vulkan library name
3.4.1. WebView Compatibility	Change for the user agent string reported by the WebView
3.7. Runtime Compatibility	Updates to memory allocation table
3.8.4. Search	Updates regarding Assistant requirements
3.8.6. Themes	Added requirement to support black system icons when requested by the SYSTEM_UI_FLAG_LIGHT_STATUS_BAR flag
3.9.1. Device Provisioning	Contains new sections for device owner provisioning and managed profile provisioning
3.9.2. Managed Profile Support	New section with requirements for device support of managed profile functionality
3.12.1. TV App	Added section to clarify TV App requirements for Android Television devices
3.12.1.1. Electronic Program Guide	Added section to clarify EPG requirements for Android Television devices
3.12.1.2. Navigation	Added section to clarify TV App navigation requirements for Android Television devices
3.12.1.3. TV input app linking	Added section to clarify TV input app linking support requirements for Android Television devices
5.1. Media Codecs	Updates regarding support for core media formats and decoding.
5.1.3. Video Codecs	Changes and additions related to Android Televisions
5.2. Video Encoding	Changes for encoders
5.3. Video Decoding	Changes for decoders, including regarding support for dynamic video resolution, frame rate switching, and more
5.4. Audio Recording	Additions related to audio capture
5.6. Audio Latency	Update regarding reporting of support for low-latency audio
5.10. Professional Audio	General updates for professional audio support; updates for mobile device (jack) specifications, USB audio host mode, and other updates
5.9. Musical Instrument Digital Interface (MIDI)	Added new section on optional Musical Instrument Digital Interface (MIDI) support
6.1. Developer Tools	Update for drivers supporting Windows 10
7.1.1.3. Screen Density	Updates for screen density, for example related to an Android watch
7.2.3. Navigation Keys	Updated requirements for device implementations that include the Assist action
7.3. Sensors (and subsections)	New requirements for some sensor types
7.3.9. High Fidelity Sensors	New section with requirements for devices supporting high fidelity sensors
7.3.10. Fingerprint	New section on requirements related to fingerprint sensors

Sensor	
7.4.2. IEEE 802.11 (Wi-Fi)	Updates regarding support for multicast DNS (mDNS)
7.4.3. Bluetooth	Addition related to Resolvable Private Address (RPA) for Bluetooth Low Energy (BLE)
7.4.4. Near-Field Communications	Additions to requirements for Near-Field Communications (NFC)
7.4.5. Minimum Network Capability	Added requirements for IPv6 support
7.6.3. Adoptable Storage	New section for implementation of adoptable storage
7.7. USB	Requirement related to implementing the AOA specification
7.8.3. Near-Ultrasound	Additions related to near-ultrasound recording, playback, and audio
8.3. Power-Saving Modes	New section with requirements regarding the App Standby and Doze modes
8.4. Power Consumption Accounting	New section with requirements for tracking hardware component power usage and attributing that power usage to specific applications
9.1. Permissions	Addition to Permissions requirements
9.7. Kernel Security Features	SE Linux updates
9.8. Privacy	Addition regarding user's consent for access to shared storage over a USB port
9.9. Full-Disk Encryption	Requirements related to full disk encryption
9.10. Verified Boot	Additional requirement for verified boot
9.11. Keys and Credentials	New section of requirements related to keys and credentials
9.12. Data Deletion	New section for "Factory Data Reset"
11. Updatable Software	Requirement related to the system update policy set by the device owner

13. Contact Us

You can join the android-compatibility forum [\[Resources, 139\]](#) and ask for clarifications or bring up any issues that you think the document does not cover.

14. Resources

1. IETF RFC2119 Requirement Levels: <http://www.ietf.org/rfc/rfc2119.txt>
2. Android Open Source Project: <http://source.android.com/>
3. Android Television features: http://developer.android.com/reference/android/content/pm/PackageManager.html#FEATURE_LEANBACK
4. Android Watch feature: http://developer.android.com/reference/android/content/res/Configuration.html#UI_MODE_TYPE_WATCH
5. Android UI_MODE_TYPE_CAR API: http://developer.android.com/reference/android/content/res/Configuration.html#UI_MODE_TYPE_CAR
6. API definitions and documentation: <http://developer.android.com/reference/packages.html>

7. Android Permissions reference: <http://developer.android.com/reference/android/Manifest.permission.html>
8. android.os.Build reference: <http://developer.android.com/reference/android/os/Build.html>
9. Android 6.0 allowed version strings: <http://source.android.com/compatibility/6.0/versions.html>
10. Android Developer Settings: <http://developer.android.com/reference/android/provider/Settings.html>
11. Telephony Provider: <http://developer.android.com/reference/android/provider/Telephony.html>
12. Android NDK ABI Management: <https://developer.android.com/ndk/guides/abis.html>
13. Advanced SIMD architecture: <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0388f/Beijfcja.html>
14. Android Extension Pack: <http://developer.android.com/guide/topics/graphics/opengl.html#aep>
15. android.webkit.WebView class: <http://developer.android.com/reference/android/webkit/WebView.html>
16. WebView compatibility: <http://www.chromium.org/>
17. HTML5: <http://html.spec.whatwg.org/multipage/>
18. HTML5 offline capabilities: <http://dev.w3.org/html5/spec/Overview.html#offline>
19. HTML5 video tag: <http://dev.w3.org/html5/spec/Overview.html#video>
20. HTML5/W3C geolocation API: <http://www.w3.org/TR/geolocation-API/>
21. HTML5/W3C webstorage API: <http://www.w3.org/TR/webstorage/>
22. HTML5/W3C IndexedDB API: <http://www.w3.org/TR/IndexedDB/>
23. Dalvik Executable Format and bytecode specification: available in the Android source code, at dalvik/docs
24. AppWidgets: http://developer.android.com/guide/practices/ui_guidelines/widget_design.html
25. Notifications: <http://developer.android.com/guide/topics/ui/notifiers/notifications.html>
26. Application Resources: <https://developer.android.com/guide/topics/resources/available-resources.html>
27. Status Bar icon style guide: <http://developer.android.com/design/style/iconography.html>
28. Notifications Resources: <https://developer.android.com/design/patterns/notifications.html>
29. Search Manager: <http://developer.android.com/reference/android/app/SearchManager.html>
30. Action Assist: http://developer.android.com/reference/android/content/Intent.html#ACTION_ASSIST
31. Android Assist APIs: <https://developer.android.com/reference/android/app/assist/package-summary.html>
32. Toasts: <http://developer.android.com/reference/android/widget/Toast.html>
33. Themes: <http://developer.android.com/guide/topics/ui/themes.html>
34. R.style class: <http://developer.android.com/reference/android/R.style.html>
35. Material design: http://developer.android.com/reference/android/R.style.html#Theme_Material
36. Live Wallpapers: <http://developer.android.com/reference/android/service/wallpaper/WallpaperService.html>
37. Overview screen resources: <http://developer.android.com/guide/components/recents.html>
38. Screen pinning: <https://developer.android.com/about/versions/android-5.0.html#ScreenPinning>
39. Input methods: <http://developer.android.com/guide/topics/text/creating-input-method.html>
40. Media Notification: <https://developer.android.com/reference/android/app/Notification.MediaStyle.html>
41. Dreams: <http://developer.android.com/reference/android/service/dreams/DreamService.html>
42. Settings.Secure LOCATION_MODE: http://developer.android.com/reference/android/provider/Settings.Secure.html#LOCATION_MODE

43. Unicode 6.1.0: <http://www.unicode.org/versions/Unicode6.1.0/>
44. Android Device Administration: <http://developer.android.com/guide/topics/admin/device-admin.html>
45. DevicePolicyManager reference: <http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html>
46. Device Owner App: [http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#isDeviceOwnerApp\(j](http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#isDeviceOwnerApp(j)
47. Android Device Owner Provisioning Flow: http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#ACTION_PROVISIONI
48. Device Owner Provisioning via NFC: https://source.android.com/devices/tech/admin/provision.html#device_owner_provisioning_via_nfc
49. Android Profile Owner App: <http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#isProfileOwnerA>
50. Android Managed Profile Provisioning flow: http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#ACTION_PROVISIONI
51. Android Accessibility Service APIs: <http://developer.android.com/reference/android/accessibilityservice/AccessibilityService.html>
52. Android Accessibility APIs: <http://developer.android.com/reference/android/view/accessibility/package-summary.html>
53. Eyes Free project: <http://code.google.com/p/eyes-free>
54. Text-To-Speech APIs: <http://developer.android.com/reference/android/speech/tts/package-summary.html>
55. Television Input Framework: <https://source.android.com/devices/tv/index.html>
56. TV App channels: <http://developer.android.com/reference/android/media/tv/TvContract.Channels.html>
57. Third-party TV inputs: https://source.android.com/devices/tv/index.html#third-party_input_example
58. TV inputs: https://source.android.com/devices/tv/index.html#tv_inputs
59. TV channel EPG fields: <https://developer.android.com/reference/android/media/tv/TvContract.Programs.html>
60. TV input app linking: http://developer.android.com/reference/android/media/tv/TvContract.Channels.html#COLUMN_APP_LINK_I
61. Reference tool documentation (for adb, aapt, ddms, systrace): <http://developer.android.com/tools/help/index.html>
62. Android apk file description: <http://developer.android.com/guide/components/fundamentals.html>
63. Manifest files: <http://developer.android.com/guide/topics/manifest/manifest-intro.html>
64. Android Media Formats: <http://developer.android.com/guide/appendix/media-formats.html>
65. Android MediaCodeList API: <http://developer.android.com/reference/android/media/MediaCodeList.html>
66. Android CamcorderProfile API: <http://developer.android.com/reference/android/media/CamcorderProfile.html>
67. WebM project: <http://www.webmproject.org/>
68. RTC Hardware Coding Requirements: <http://www.webmproject.org/hardware/rtc-coding-requirements/>
69. AudioEffect API: <http://developer.android.com/reference/android/media/audiofx/AudioEffect.html>
70. Android android.content.pm.PackageManager class and Hardware Features List: <http://developer.android.com/reference/android/content/pm/PackageManager.html>
71. HTTP Live Streaming Draft Protocol: <http://tools.ietf.org/html/draft-pantos-http-live-streaming-03>
72. ADB: <http://developer.android.com/tools/help/adb.html>

73. Dumpsys: <https://source.android.com/devices/input/diagnostics.html>
74. DDMS: <http://developer.android.com/tools/debugging/ddms.html>
75. Monkey testing tool: <http://developer.android.com/tools/help/monkey.html>
76. SysyTrace tool: <http://developer.android.com/tools/help/systrace.html>
77. Android Application Development-Related Settings: http://developer.android.com/reference/android/provider/Settings.html#ACTION_APPLICATION_DEVELOPI
78. Supporting Multiple Screens: http://developer.android.com/guide/practices/screens_support.html
79. android.util.DisplayMetrics: <http://developer.android.com/reference/android/util/DisplayMetrics.html>
80. RenderScript: <http://developer.android.com/guide/topics/renderscript/>
81. Android extension pack for OpenGL ES: <https://developer.android.com/reference/android/opengl/GLES31Ext.html>
82. Hardware Acceleration: <http://developer.android.com/guide/topics/graphics/hardware-accel.html>
83. EGL Extension-EGL_ANDROID_RECORDABLE: http://www.khronos.org/registry/egl/extensions/ANDROID/EGL_ANDROID_recordable.txt
84. Display Manager: <http://developer.android.com/reference/android/hardware/display/DisplayManager.html>
85. android.content.res.Configuration: <http://developer.android.com/reference/android/content/res/Configuration.html>
86. Touch Input Configuration: <http://source.android.com/devices/tech/input/touch-devices.html>
87. Motion Event API: <http://developer.android.com/reference/android/view/MotionEvent.html>
88. Key Event API: <http://developer.android.com/reference/android/view/KeyEvent.html>
89. Android Open Source sensors: <http://source.android.com/devices/sensors>
90. android.hardware.SensorEvent: <http://developer.android.com/reference/android/hardware/SensorEvent.html>
91. Timestamp sensor event: <http://developer.android.com/reference/android/hardware/SensorEvent.html#timestamp>
92. Android Open Source composite sensors: https://source.android.com/devices/sensors/sensor-types.html#composite_sensor_type_summary
93. Continuous trigger mode: <https://source.android.com/devices/sensors/report-modes.html#continuous>
94. Accelerometer sensor: http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ACCELEROMETER
95. Android Fingerprint API: <https://developer.android.com/reference/android/hardware/fingerprint/package-summary.html>
96. Android Fingerprint HAL: <https://source.android.com/devices/tech/security/authentication/fingerprint-hal.html>
97. Wi-Fi Multicast API: <http://developer.android.com/reference/android/net/wifi/WifiManager.MulticastLock.html>
98. Wi-Fi Direct (Wi-Fi P2P): <http://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html>
99. WifiManager API: <http://developer.android.com/reference/android/net/wifi/WifiManager.html>
100. Bluetooth API: <http://developer.android.com/reference/android/bluetooth/package-summary.html>
101. Bluetooth ScanFilter API: <https://developer.android.com/reference/android/bluetooth/le/ScanFilter.html>
102. NFC Barcode: <http://developer.android.com/reference/android/nfc/tech/NfcBarcode.html>
103. NDEF Push Protocol: <http://source.android.com/compatibility/ndef-push-protocol.pdf>
104. Android Beam: <http://developer.android.com/guide/topics/connectivity/nfc/nfc.html>

105. Android NFC Sharing Settings: http://developer.android.com/reference/android/provider/Settings.html#ACTION_NFCSHARING_SETTINGS
106. NFC Connection Handover: http://members.nfc-forum.org/specs/spec_list/#conn_handover
107. Bluetooth Secure Simple Pairing Using NFC: http://members.nfc-forum.org/apps/group_public/download.php/18688/NFCForum-AD-BTSSP_1_1.pdf
108. Host-based Card Emulation: <http://developer.android.com/guide/topics/connectivity/nfc/hce.html>
109. Content Resolver: <http://developer.android.com/reference/android/content/ContentResolver.html>
110. Camera orientation API: [http://developer.android.com/reference/android/hardware/Camera.html#setDisplayOrientation\(int\)](http://developer.android.com/reference/android/hardware/Camera.html#setDisplayOrientation(int))
111. Camera: <http://developer.android.com/reference/android/hardware/Camera.html>
112. Camera: <http://developer.android.com/reference/android/hardware/Camera.Parameters.html>
113. Camera hardware level: https://developer.android.com/reference/android/hardware/camera2/CameraCharacteristics.html#INFO_SUF
114. Camera version support: <http://source.android.com/devices/camera/versioning.html>
115. Android DownloadManager: <http://developer.android.com/reference/android/app/DownloadManager.html>
116. Android File Transfer: <http://www.android.com/filetransfer>
117. Adoptable storage: <http://source.android.com/devices/storage/adoptable.html>
118. Android Open Accessories: <http://developer.android.com/guide/topics/connectivity/usb/accessory.html>
119. Android USB Audio: http://developer.android.com/reference/android/hardware/usb/UsbConstants.html#USB_CLASS_AUDIO
120. USB Charging Specification: http://www.usb.org/developers/docs/devclass_docs/USB_Battery_Charging_1.2.pdf
121. USB Host API: <http://developer.android.com/guide/topics/connectivity/usb/host.html>
122. Wired audio headset: <http://source.android.com/accessories/headset-spec.html>
123. Power profile components: <http://source.android.com/devices/tech/power/values.html>
124. Battery stats: <http://source.android.com/devices/tech/power/batterystats.html>
125. Power usage summary: http://developer.android.com/reference/android/content/Intent.html#ACTION_POWER_USAGE_SUMMARY
126. Android Security and Permissions reference: <http://developer.android.com/guide/topics/security/permissions.html>
127. UserManager reference: <http://developer.android.com/reference/android/os/UserManager.html>
128. External Storage reference: <http://source.android.com/devices/storage>
129. External Storage APIs: <http://developer.android.com/reference/android/os/Environment.html>
130. SMS Short Code: http://en.wikipedia.org/wiki/Short_code
131. Secure lock screen reporting: [http://developer.android.com/reference/android/app/KeyguardManager.html#isDeviceSecure\(\)](http://developer.android.com/reference/android/app/KeyguardManager.html#isDeviceSecure())
132. Android Open Source Encryption: <http://source.android.com/devices/tech/security/encryption/index.html>
133. Android Keystore System: <https://developer.android.com/training/articles/keystore.html>
134. KeyChain API: <https://developer.android.com/reference/android/security/KeyChain.html>
135. Keystore API: <https://developer.android.com/reference/java/security/KeyStore.html>
136. Gatekeeper HAL: <http://source.android.com/devices/tech/security/authentication/gatekeeper.html>
137. Android Compatibility Program Overview: <http://source.android.com/compatibility/index.html>
138. SystemUpdatePolicy class:

<http://developer.android.com/reference/android/app/admin/SystemUpdatePolicy.html>

139. Android Compatibility forum: <https://groups.google.com/forum/#!forum/android-compatibility>

140. Handling app links: <https://developer.android.com/training/app-links/index.html>

141. Google Digital Asset Links: <https://developers.google.com/digital-asset-links>

Many of these resources 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 references included above are considered by inclusion to be part of this Compatibility Definition.