android

Compatibility Program

Android 5.0 Compatibility Definition

Last updated: January 11, 2015

Table of Contents

1. Introduction

This document enumerates the requirements that must be met in order for devices to be compatible with Android 5.0.

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 5.0. A "device implementation" or "implementation" is the hardware/software solution so developed.

To be considered compatible with Android 5.0, 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 encouraged 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 this practice is strongly discouraged, 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]

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 5.0 compatible, unless the requirement is explicitly described to be only applicable to a specific Android device type.

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	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	SHOULD
Sensors	Accelerometer	7.3.1 Accelerometer	SHOULD		SHOULD	SHOULD
	GPS	7.3.3. GPS	SHOULD			
Connectivity	Wi-Fi	7.4.2. IEEE 802.11	SHOULD	MUST		SHOULD
	Wi-Fi Direct	7.4.2.1. Wi-Fi Direct	SHOULD	SHOULD		SHOULD
	Bluetooth	7.4.3. Bluetooth	SHOULD	MUST	MUST	SHOULD
	Bluetooth Low Energy	7.4.3. Bluetooth	SHOULD	MUST	SHOULD	SHOULD
	USB peripheral/ host mode	7.7. USB	SHOULD			SHOULD
Output	Speaker and/or Audio output ports	7.8.2. Audio Output	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, 5] 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, 6]. 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, 7] 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, 8].
VERSION.SDK	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 5.0, this field MUST have the integer value 21.

VERSION.SDK_INT	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 5.0, this field MUST have the integer value 21.
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	A string that uniquely identifies this build. It SHOULD be reasonably human-readable. It MUST follow this template:
	`$(BRAND)/$(PRODUCT)/$(DEVICE):$(VERSION.RELEASE)/$(ID)/$(VERSION.INCREMENTAL):$(TYPE)/$(TAGS)`
	For example:
	`acme/myproduct/mydevice:5.0/LRWXX/3359:userdebug/test-keys`
	The fingerprint MUST NOT include whitespace characters. If other fields included in the template above have whitespace characters, they MUST be replaced in the build fingerprint with another character, such as the underscore ("_") character. The value of this field MUST be encodable as 7-bit ASCII.

HARDWARE	The name of the hardware (from the kernel command line or /proc). It SHOULD be reasonably human-readable. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression `"^[a-zA-Z0-9_-]+$"`.
HOST	A string that uniquely identifies the host the build was built on, in human-readable format. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
ID	An identifier chosen by the device implementer to refer to a specific release, in human-readable format. This field can be the same as `android.os.Build.VERSION.INCREMENTAL`, but SHOULD be a value sufficiently meaningful for end users to distinguish between software builds. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression `"^[a-zA-Z0-9._-]+$"`.
MANUFACTURER	The trade name of the Original Equipment Manufacturer (OEM) of the product. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
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. 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 ("").

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 Overrides

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.

However, device implementations MAY provide default activities for specific URI patterns (eg. http://play.google.com) if the default activity provides a more specific filter for the data URI. For example, an intent filter specifying the data URI "http://www.android.com" is more specific than the browser filter for "http://". Device implementations MUST provide a user interface for users to modify the default activity for intents.

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, 9]
- 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 in the latest version of the Android NDK, "NDK Programmer's Guide | ABI Management" in `docs/` directory
- 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, 11] 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.

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

3.4. Web Compatibility

3.4.1. WebView Compatibility

The complete implementation of the `android.webkit.Webview` API MAY be provided on Android Watch devices but MUST be provided on all other types of device implementations.

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, 12]. 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 5.0. This build includes a specific set of functionality and security fixes for the WebView [Resources, 13].
- The user agent string reported by the WebView MUST be in this format:

```
Mozilla/5.0 (Linux; Android $(VERSION); $(MODEL) Build/$(BUILD)) 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, 14].

3.4.2. Browser Compatibility

Android Television and Watch Devices 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, 14] as possible. Minimally, device implementations MUST support each of these APIs associated with HTML5:

- application cache/offline operation [Resources, 15]
- the `<video>` tag [Resources, 16]
- geolocation [Resources, 17]

Additionally, device implementations MUST support the HTML5/W3C webstorage API [Resources, 18], and SHOULD support the HTML5/W3C IndexedDB API [Resources, 19]. 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 `<uses-library>` mechanism) are affected by the increased memory usage of such APIs.

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

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

3.7. Runtime Compatibility

Device implementations MUST support the full Dalvik Executable (DEX) format and Dalvik bytecode specification and semantics [Resources, 20]. 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
small / normal	120 dpi (ldpi)	16MB
	160 dpi (mdpi)	
	213 dpi (tvdpi)	32MB
	240 dpi (hdpi)	
	320 dpi (xhdpi)	64MB
	400 dpi (400dpi)	96MB
	480 dpi (xxhdpi)	128MB
	560 dpi (560dpi)	192MB
	640 dpi (xxxhdpi)	256MB
large	120 dpi (ldpi)	16MB
	160 dpi (mdpi)	32MB
	213 dpi (tvdpi)	64MB
	240 dpi (hdpi)	
	320 dpi (xhdpi)	128MB
	400 dpi (400dpi)	192MB
	480 dpi (xxhdpi)	256MB
	560 dpi (560dpi)	384MB
	640 dpi (xxxhdpi)	512MB

xlarge	160 dpi (mdpi)	64MB
	213 dpi (tvdpi)	96MB
	240 dpi (hdpi)	
	320 dpi (xhdpi)	192MB
	400 dpi (400dpi)	288MB
	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, 21] 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, 21] 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, 22], 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, sound files, etc.) provided for in the APIs [Resources, 23], or in the Status/System Bar icon style guide [Resources, 24]. 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.

Device implementations MUST properly display and execute these notifications, including the title/name, icon, text as documented in the Android APIs [Resources, 25].

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, 26] 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.

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, 27]. 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, 28]. Device implementations MUST NOT alter any of the Holo theme attributes exposed to applications [Resources, 29].

Android 5.0 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, 30].

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, 29].

Android supports a new 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 [Resources, 29].

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, 31]. 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, 32], 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, 33] 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 ENCOURAGED 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, 34]. 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, 35]. Device implementations that support a lock screen in the device MUST support the Media Notification Template along with other notifications.

3.8.11. Dreams

Android includes support for interactive screensavers called Dreams [Resources, 36]. 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, 37].

3.8.13. Unicode and Font

Android includes support for color emoji characters. When Android device implementations include an IME, devices MUST provide an input method to the user for the Emoji characters defined in Unicode 6.1 [Resources, 38]. All devices MUST be capable of rendering these emoji characters in color glyph.

Android 5.0 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, 39]. Device implementations MUST provide an implementation of the `DevicePolicyManager` class [Resources, 40]. Device implementations that include support for lock screen MUST support the full range of device administration policies defined in the Android SDK documentation [Resources, 39] and report the platform feature `android.software.device_admin`.

Device implementations MAY have a preinstalled application performing device administration functions but this application MUST NOT be set out-of-the box as the default Device Owner app [Resources, 41].

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, 42]. Device implementations MUST provide an implementation of the Android accessibility framework consistent with the default Android implementation. Device implementations MUST meet the following requirements:

- MUST support third-party accessibility service implementations through the `android.accessibilityservice` APIs [Resources, 43]
- MUST generate `AccessibilityEvents` and deliver these events to all registered `AccessibilityService` implementations in a manner consistent with the default Android implementation

- Unless an Android Watch device with no audio output, device implementations 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, 44].

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, 45]. Device implementations reporting the feature `android.hardware.audio.output` MUST meet these requirements related to the Android TTS framework.

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 Television Input Framework [Resources, 46].

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

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, 47].

Devices implementations MUST NOT extend either the .apk [Resources, 48], Android Manifest [Resources, 49], Dalvik bytecode [Resources, 20], 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, 50] 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. 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 Type(s) / 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.	• 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 @ 8kHz	3GPP (.3gp)
AMR-WB	REQUIRED[3]	REQUIRED[3]	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz	
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	• Type 0 and 1 (.mid, .xmf, .mxmf) • RTTTL/RTX (.rtttl, .rtx) • OTA (.ota) • iMelody (.imy)
Vorbis		REQUIRED		• 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 Type(s) / 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

Video codecs are optional for Android Watch device implementations.

Format / Codec	Encoder	Decoder	Details	Supported File Type(s) / Container Formats
H.263	REQUIRED[1]	REQUIRED[2]		• 3GPP (.3gp) • MPEG-4 (.mp4)
H.264 AVC	REQUIRED[2]	REQUIRED[2]	See section 5.2 and 5.3 for details	• 3GPP (.3gp) • MPEG-4 (.mp4) • MPEG-TS (.ts, AAC audio only, not seekable, Android 3.0+)
H.265 HEVC		REQUIRED[2]	See section 5.3 for details	MPEG-4 (.mp4)
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	• WebM (.webm) [Resources, 110] • Matroska (.mkv, Android 4.0+)[4]
VP9		REQUIRED[2] (Android 4.4+)	See section 5.3 for details	• WebM (.webm) [Resources, 110] • 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, 51].

[4] Device implementations SHOULD support writing Matroska WebM files.

5.2. Video Encoding

Video codecs are optional for Android Watch device implementations.

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 switching within the same stream for VP8, VP9 ,H.264, and H.265 codecs.

Android device implementations with H.264 decoders, MUST support Baseline Profile Level 3 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	30 fps / 60 fps[2]	30 fps / 60 fps[2]
Video bitrate	800 Kbps	2 Mbps	8 Mbps	20 Mbps

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

	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	30 fps	30 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 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	30 fps	30 fps
Video bitrate	600 Kbps	1.6 Mbps	4 Mbps	10 Mbps	20 Mbps

[1] Required for Android Television device implementation, but for other type of devices only when supported by hardware.
[2] Required for 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 **very strongly encouraged** 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

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

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 1Khz 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, 52]. 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 5 milliseconds.
- **OpenSL ES PCM buffer queue API**—The set of PCM-related OpenSL ES APIs within Android NDK; see `NDK_root/docs/opensles/index.html`.

Device implementations that declare `android.hardware.audio.output` SHOULD 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 MAY report support for low-latency audio, by reporting the feature `android.hardware.audio.low_latency` via the `android.content.pm.PackageManager` class [Resources, 53]. 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` SHOULD 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, 50]. 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, 54]

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.

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, 55]

  Device implementations MUST support all `adb` functions as documented in the Android SDK including dumpsys [Resources, 56]. 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, 57]

  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, 58]

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

- **SysTrace** [Resources, 59]

  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 9 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, 60]. 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's 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, 53]

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, 61]. 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 longer dimension of the screen to the shorter dimension. For example, a display of 480x854 pixels would be 854 / 480 = 1.779, or roughly "16:9".
- **density-independent pixel (dp)**—The virtual pixel unit normalized to a 160 dpi screen, calculated as: pixels = dps * (density / 160).

7.1.1. Screen Configuration

7.1.1.1. Screen Size

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, 61] 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 at any time for the default display.

- 120 dpi (ldpi)

- 160 dpi (mdpi)
- 213 dpi (tvdpi)
- 240 dpi (hdpi)
- 320 dpi (xhdpi)
- 400 dpi (400dpi)
- 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, 62] 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, 63].

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 `GLES10.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, 64] 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, 65].

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, 65].

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, 66].

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. 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. External 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, 67].

7.2. Input Devices

7.2.1. Keyboard

Android Watch devices MAY but other type of device implementations MUST implement a soft keyboard.

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, 68] (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, 68]
- 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.
- 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 5.0 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 5.0, 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` <= 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 supports Assist action [Resources, 69]. Android device implementations except for Android Watch devices MUST make the Assist action available to the user at all times when running applications. The Assist action SHOULD be implemented as a long-press on the Home button or a swipe-up gesture on the software Home key. This function MAY be implemented via another physical button, software key, or gesture, but MUST be accessible with a single action (e.g. tap, double-click, or gesture) when other navigation keys are visible.

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, 68] 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, 70] 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 5.0 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, 71]
- 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, 71]
- 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, 71]
- 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, 72]
[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, 71]

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, 71]

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, 72].

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, 73]. For example, device implementations:

- MUST accurately report the presence or absence of sensors per the `android.content.pm.PackageManager` class [Resources, 53]
- 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, 74]
- 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 **very strongly encouraged** 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, 75].

The list above is not comprehensive; the documented behavior of the Android SDK and the Android Open Source Documentations on Sensors [Resources, 73] 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, 76]. 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, 76].

Some Android sensor supports a "continuous" trigger mode, which returns data continuously [Resources, 77]. 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 encouraged to include this sensor. If a device implementation does include a 3-axis accelerometer, it:

- MUST implement and report `TYPE_ACCELEROMETER` sensor [Resources, 78]
- MUST be able to report events up to a frequency of at least 100 Hz and SHOULD report events up to at least 200 Hz
- MUST comply with the Android sensor coordinate system as detailed in the Android APIs [Resources, 74]
- 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 8-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/s^, where the standard deviation should be calculated on a per axis basis on samples collected over a period of at least 3 seconds at the fastest sampling rate
- 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 **very strongly encouraged** 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 encouraged to implement the `TYPE_GAME_ROTATION_VECTOR` sensor.
- SHOULD 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 encouraged 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, 74]
- MUST be capable of measuring between -900 μT and +900 μT on each axis before saturating
- MUST have a hard iron offset value less than 700 μT and SHOULD have a value below 200 μT, by placing the magnetometer far from dynamic (current-induced) and static (magnet-induced) magnetic fields
- MUST have a resolution equal or denser than 0.6 μT and SHOULD have a resolution equal or denser than 0.2 μT
- 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 μT
- SHOULD 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 encouraged 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 100 Hz and 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 rad^2 / s^2 per Hz (variance per Hz, or rad^2 / s). The variance is allowed to vary with the sampling rate, but must be constrained by this value. In other words, if you measure the variance of the gyro at 1 Hz sampling rate it should be no greater than 1e-7 rad^2/s^2.
- SHOULD 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 encouraged 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

7.4. 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, 79]
- MUST support multicast DNS (mDNS) and MUST NOT filter mDNS packets (224.0.0.251) at any time of operation including when the screen is not in an active state

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, 80]. 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, 81]. 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 Television device implementations MUST support Bluetooth and Bluetooth LE and Android Watch device implementations MUST support Bluetooth.

Android includes support for Bluetooth and Bluetooth Low Energy [Resources, 82]. 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, 82]
- SHOULD support offloading of the filtering logic to the bluetooth chipset when implementing the ScanFilter API [Resources, 83], 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.BluetoothAdapater.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, 53]
- 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 6319-4)
        - IsoDep (ISO 14443-4)
        - NFC Forum Tag Types 1, 2, 3, 4 (defined by the NFC Forum)
- SHOULD be capable of reading and writing NDEF messages via the following NFC standards. Note that while the NFC standards below are stated as SHOULD, 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)
- MUST be capable of transmitting and receiving data via the following peer-to-peer standards and protocols:
    - ISO 18092
    - LLCP 1.0 (defined by the NFC Forum)
    - SDP 1.0 (defined by the NFC Forum)
    - NDEF Push Protocol [Resources, 84]
    - SNEP 1.0 (defined by the NFC Forum)
- MUST include support for Android Beam [Resources, 85]:
    - 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, 86]
    - 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, 87] and "Bluetooth Secure Simple Pairing Using NFC version 1.0" [Resources, 88] 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 5.0 includes support for NFC Host Card Emulation (HCE) mode. If a device implementation does include an NFC controller 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, 10]

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, 53]. Note that this is not a standard Android feature and as such does not appear as a constant on the 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, 53], 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.

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, 89].

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 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, 90] 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, 91], 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, 92].

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, 93] and report the appropriate framework feature flags [Resources, 94].

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, 94]; 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
xhdpi or lower on small/normal screens hdpi or lower on large screens mdpi or lower on extra large screens	512MB	832MB
400dpi or higher on small/normal screens xhdpi or higher on large screens tvdpi or higher on extra large screens	896MB	1280MB
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.

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 **very strongly encouraged** 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, 95]. 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, device implementations MUST provide some mechanism to access the contents of shared storage from a host computer, such as USB mass storage (UMS) or Media Transfer Protocol (MTP). Device implementations MAY use USB mass storage, but SHOULD use Media Transfer Protocol. If the device implementation supports Media Transfer Protocol, it:

- SHOULD be compatible with the reference Android MTP host, Android File Transfer [Resources, 96]
- SHOULD report a USB device class of 0x00
- SHOULD report a USB interface name of 'MTP'

If the device implementation lacks USB ports, it MUST provide a host computer with access to the contents of shared storage by some other means, such as a network file system.

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 **very strongly encouraged 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 **very strongly encouraged 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, if the device reports the `android.hardware.feature.output` feature or the `android.hardware.camera` feature.
- 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, 97]
    - MUST implement the USB audio class as documented in the Android SDK documentation [Resources, 98]
- It SHOULD implement support to draw 1.5 A current during HS chirp and traffic as specified in the USB battery charging specification [Resources, 99]. Existing and new Android devices are **very strongly encouraged 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 **very strongly RECOMMENDED** to implement the USB audio class as documented in the Android SDK documentation [Resources, 98]
- 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, 100]
- SHOULD support the Charging Downstream Port output current range of 1.5 A ~ 5 A as specified in the USB Battery Charging Specifications [Resources, 99]

7.8. Audio

7.8.1. Microphone

Android Handheld and Watch devices 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

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

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, 101], 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

8. Performance Compatibility

Some minimum performance criterias are critical to the user experience and impacts 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 file access performance consistency for read and write operations.

- **Sequential write**—Device implementations MUST ensure a sequential write performance of 10MB/s for a 256MB file using 10MB write buffer.
- **Random write**—Device implementations MUST ensure a random write performance of 0.5MB/s for a 256MB file using 4KB write buffer.
- **Sequential read**—Device implementations MUST ensure a sequential read performance of 15MB/s for a 256MB file using 10MB write buffer.
- **Random read**—Device implementations MUST ensure a random read performance of 3.5MB/s for a 256MB file using 4KB write buffer.

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, 102] 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, 102]. 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.

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, 102].

9.3. Filesystem Permissions

Device implementations MUST support the Android file access permissions model as defined in the Security and Permissions reference [Resources, 102].

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, 103]. Device implementations MAY enable multiple users, but when enabled MUST meet the following requirements related to multi-user support [Resources, 104]:

- 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, 102]
- Device implementations MAY support creating users and managed profiles via the `android.app.admin.DevicePolicyManager` APIs, and if supported, MUST declare the platform feature flag `android.software.managed_users`.
- Device implementations that declare the feature flag `android.software.managed_users` MUST use the upstream AOSP icon badge to represent the managed applications and other badge UI elements like Recents & Notifications.

- 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, 105] for primary external storage.

9.6. Premium SMS Warning

Android includes support for warning users of any outgoing premium SMS message [Resources, 106] . 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 can 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, if 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 an equivalent mandatory access control system if using a kernel other than Linux and meet the following requirements, which are satisfied by the reference implementation in the upstream Android Open Source Project.

Device implementations:

- MUST support a SELinux policy that allows the SELinux mode to be set on a per-domain basis, and MUST configure all domains in enforcing mode. No permissive mode domains are allowed, including domains specific to a `device/vendor`
- SHOULD load policy from `/sepolicy` file on the device

- MUST NOT modify, omit, or replace the `neverallow` rules present within the `sepolicy` file provided in the upstream Android Open Source Project (AOSP) and the policy MUST compile with all `neverallow` present, for both AOSP SELinux domains as well as `device/vendor` specific domains
- MUST support dynamic updates of the SELinux policy file without requiring a system image update

Device implementations SHOULD retain the default SELinux policy provided in the upstream Android Open Source Project, until they have first audited their additions to the SELinux policy. 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.

9.9. Full-Disk Encryption

Optional for Android device implementations without a lock screen.

If the device implementation has a lock screen, the device MUST support full-disk encryption of the application private data, (`/data` partition) as well as the SD card partition if it is a permanent, non-removable part of the device [Resources, 107]. For devices supporting full-disk encryption, the full-disk encryption SHOULD be enabled all the time after the user has completed the out-of-box experience. While this requirement is stated as SHOULD for this version of the Android platform, it is **very strongly RECOMMENDED** as we expect this to change to MUST in the future versions of Android. 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

Device implementations SHOULD support verified boot for device integrity, and if the feature is supported it MUST declare the platform feature flag `android.software.verified_boot`. While this requirement is stated as SHOULD for this version of the Android platform, it is **very strongly RECOMMENDED** as we expect this to change to MUST in the future versions of Android. The upstream Android Open Source Project provides a preferred implementation of this feature based on the linux kernel feature `dm-verity`.

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 **very strongly encouraged** 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, 108] 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 5.0. 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, the device MUST support Over-the-air download 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 5.0 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 5.0, 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.

12. Document Changelog

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

Section(s)	Summary of change
1. Introduction	Updated requirements to refer to SDK documentation as source of truth.
2. Device Types	Included definitions for device types for handheld, television, and watch devices.
2.1 Device Configuration	Added non-exhaustive list to illustrate hardware configuration deviation across devices.
3.1. Managed API Compatibility	MUST also provide complete implementations of APIs with "@SystemApi" marker in the upstream Android source code.
3.2.2. Build Parameters	Included SUPPORTED_ABIS, SUPPORTED_32_BIT_ABIS, and SUPPORTED_64_BIT_ABIS parameters in list, updated PRODUCT to require unique Product SKUs, and updated TAGS.
3.2.3.1. Core Application Intents	Clarified language that the compatibility requirement is for mainly the intents pattern
3.2.3.5. Default App Settings	Included new requirements for home screen, NFC, and default SMS applications.
3.3.1 Application Binary Interfaces	Added requirements to support equivalent 32-bit ABI if any 64-bit ABI is supported. Updated parameters to reflect this change.
3.4.1. WebView Compatibility	Webview compatibility required for all devices except Android Watch devices. Removed Locale string requirement.

3.4.2. Browser compatibility	Android Television and Watch Devices MAY omit a browser application, but all other types of device implementations MUST include one.
3.7. Runtime compatibility	Updated Minimum application memory requirements
3.8.2. Widgets	Widget support is optional for all device types, but recommended for Handheld Devices.
3.8.3. Notifications	Expanded definitions for types of supported notifications.
3.8.4. Search	Android Television devices MUST include global search. All other device types SHOULD.
3.8.6. Themes	Devices MUST support material theme.
3.8.7. Live Wallpapers	Devices that include live wallpaper MUST report the platform feature flag android.software.live_wallpaper
3.8.8. Activity Switching	Advised requirement to support new Recents User Interface
3.8.10. Lock Screen Media Remote Control	Remote Control Client API deprecated in favor of the Media Notification Template
3.8.11. Dreams	Optional for Android Watch devices. Required for all other device types.
3.8.13 Unicode and font	MUST support Roboto 2 in addition to existing requirements.
3.12. TV Input Framework	Android Television device implementations MUST support Television Input Framework.
5.1. Media Codecs	Added 3 sections for Audio, Image, and Video codecs.
5.4 Audio Recording	Broken into subsections
5.4.1. Raw audio capture	Defined characteristics for raw audio capture on devices that declare `android.hardware.microphone`
5.5. Audio Playback	Added section 5.5. Audio Playback with 2 subsections: 5.5.1 Audio Effects and 5.5.2. Audio Output Volume
5.6 Audio Latency	Added definitions and requirements for cold output jitter, cold input jitter, and continuous round-trip latency.
5.8 Secure Media	Included secure media requirements from 7.1.8. External Displays and added requirements for Android Television.
6.1. Developer Tools	Updated resources.
6.2.1. Experimental	Removed section
7. Hardware Compatibility	Updated to reflect that device implementations MUST consistently report accurate hardware configuration for the same build fingerprint.
7.1.1.1. Screen Size	Updated to reflect Android Watch devices screen size and that the value can't

	change
7.1.1.2. Screen Aspect Ratio	Updated to reflect Android Watch devices screen aspect ratio (1:1).
7.1.3. Screen Orientation	Updated to reflect that devices with a fixed orientation landscape screen SHOULD only report that orientation.
7.1.4. 2D and 3D Graphics Acceleration	Added that Android devices MAY support the Android extension pack.
(old) 7.1.6. Screen Types	Section Removed
7.1.6. Screen Technology	Updated pixel aspect ratio (PAR) to be between 0.9 and 1.15. (~15% tolerance)
7.1.7. External Displays	Moved part of section to section 5.8. Secure Media.
7.2.2. Non-touch Navigation	Android Television devices MUST support D-pad.
7.2.3. Navigation keys	Included language for support across different device types.
7.2.4. Touchscreen input	Android Watch devices MUST support touchscreen input.
7.2.6. Game Controller Support	Added section with Android Television requirements.
7.2.7. Remote Control	Added section with Android Television requirements.
7.3. Sensors	Redefined synthetic sensors as composite sensors and streaming sensors as continuous sensors. Sensors should report event time in nanoseconds.
7.3.1. Accelerometer	Clarified required sensor types and revised requirement thresholds.
7.3.2. Magnetometer	Clarified required sensor types and revised requirement thresholds.
7.3.4. Gyroscope	Clarified required sensor types and revised requirement thresholds.
7.3.5. Barometer	Changed from MAY to SHOULD implement barometer. MUST implement and report `TYPE_PRESSURE` sensor.
7.3.6. Thermometer	Devices MAY include ambient thermometer. MAY but SHOULD NOT include CPU thermometer.
7.3.8. 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.
7.4.2. IEEE 802.11 (Wi-Fi)	Android Television devices MUST include Wi-Fi support. Devices that DO support wifi must report `android.hardware.wifi`.
7.4.2.1. Wi-Fi Direct	MUST report the hardware feature `android.hardware.wifi.direct`.
7.4.2.2. Wi-Fi Tunneled Direct Link Setup	Android Television devices MUST include support for Wi-Fi TDLS.
7.5. Cameras	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.
7.5.3. External Cameras	Added requirements that device implementations with USB host mode MAY include support for an external camera.
7.5.5. Camera System Features	Added list of camera features and when they should be defined.
7.6.1. Minimum Memory and Storage	Updated requirements for 32- and 64-bit devices. SVELTE memory requirement removed. Devices MUST have at least 1.5GB of non-volatile storage
7.6.2. Application Shared Storage	Updated requirements for user-accessible removable storage
7.7. USB	Removed requirements for non-charging ports being on the same edge as the micro-USB port. Updated requirements for Host and Peripheral mode.
7.8.1. Audio	Moved microphone section here. Added requirements for Audio Output and Audio Analog ports.
8. Performance Compatibility	Added requirements for user interface consistency.
9.5. Multi-User Support	Multi-user support feature is optional for all device types. Detailed requirements by device type in section.
9.7. Kernel Security Features	MAY have a visible user interface when an unblocked security violation occurs resulting in a successful exploit. No permissive mode domains allowed.
9.9. Full-Disk Encryption	Devices with a lock screen MUST support full-disk encryption. For new devices, full-disk encryption must be enabled out of box.
9.10 Verified boot	Added section to recommend that Device implementations support verified boot for device integrity.
10.3. Reference Applications	Removed section from CDD.
11. Updatable Software	If a device supports 802.11 or Bluetooth PAN (Personal Area Network) profile, then it MUST support Over-the-air download with offline update via reboot.
14. Resources	Resources moved from section 2 to section 14

13. Contact Us

You can join the android-compatibility forum [Resources, 109] 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. API definitions and documentation: http://developer.android.com/reference/packages.html
6. Android Permissions reference: http://developer.android.com/reference/android/Manifest.permission.html
7. `android.os.Build` reference: http://developer.android.com/reference/android/os/Build.html
8. Android 5.0 allowed version strings: http://source.android.com/compatibility/5.0/versions.html
9. Telephony Provider: http://developer.android.com/reference/android/provider/Telephony.html
10. Host-based Card Emulation: http://developer.android.com/guide/topics/connectivity/nfc/hce.html
11. Android Extension Pack: http://developer.android.com/guide/topics/graphics/opengl.html#aep
12. `android.webkit.WebView` class: http://developer.android.com/reference/android/webkit/WebView.html
13. WebView compatibility: http://www.chromium.org/
14. HTML5: http://www.whatwg.org/specs/web-apps/current-work/multipage/
15. HTML5 offline capabilities: http://dev.w3.org/html5/spec/Overview.html#offline
16. HTML5 video tag: http://dev.w3.org/html5/spec/Overview.html#video
17. HTML5/W3C geolocation API: http://www.w3.org/TR/geolocation-API/
18. HTML5/W3C webstorage API: http://www.w3.org/TR/webstorage/
19. HTML5/W3C IndexedDB API: http://www.w3.org/TR/IndexedDB/
20. Dalvik Executable Format and bytecode specification: available in the Android source code, at dalvik/docs
21. AppWidgets: http://developer.android.com/guide/practices/ui_guidelines/widget_design.html
22. Notifications: http://developer.android.com/guide/topics/ui/notifiers/notifications.html
23. Application Resources: https://developer.android.com/guide/topics/resources/available-resources.html
24. Status Bar icon style guide: http://developer.android.com/design/style/iconography.html
25. Notifications Resources: https://developer.android.com/design/patterns/notifications.html
26. Search Manager: http://developer.android.com/reference/android/app/SearchManager.html
27. Toasts: http://developer.android.com/reference/android/widget/Toast.html
28. Themes: http://developer.android.com/guide/topics/ui/themes.html
29. R.style class: http://developer.android.com/reference/android/R.style.html
30. Material design: http://developer.android.com/reference/android/R.style.html#Theme_Material
31. Live Wallpapers: http://developer.android.com/reference/android/service/wallpaper/WallpaperService.html
32. Overview screen resources: http://developer.android.com/guide/components/recents.html
33. Screen pinning: https://developer.android.com/about/versions/android-5.0.html#ScreenPinning
34. Input methods: http://developer.android.com/guide/topics/text/creating-input-method.html
35. Media Notification: https://developer.android.com/reference/android/app/Notification.MediaStyle.html
36. Dreams: http://developer.android.com/reference/android/service/dreams/DreamService.html
37. `Settings.Secure LOCATION_MODE`:
http://developer.android.com/reference/android/provider/Settings.Secure.html#LOCATION_MODE
38. Unicode 6.1.0: http://www.unicode.org/versions/Unicode6.1.0/
39. Android Device Administration: http://developer.android.com/guide/topics/admin/device-admin.html

40. DevicePolicyManager reference:
http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html
41. Android Device Owner App:
http://developer.android.com/reference/android/app/admin/DevicePolicyManager.html#isDeviceOwnerApp(java.lang.String)
42. Android Accessibility Service APIs:
http://developer.android.com/reference/android/accessibilityservice/AccessibilityService.html
43. Android Accessibility APIs:
http://developer.android.com/reference/android/view/accessibility/package-summary.html
44. Eyes Free project: http://code.google.com/p/eyes-free
45. Text-To-Speech APIs: http://developer.android.com/reference/android/speech/tts/package-summary.html
46. Television Input Framework: https://source.android.com/devices/tv/index.html
47. Reference tool documentation (for adb, aapt, ddms, systrace):
http://developer.android.com/guide/developing/tools/index.html
48. Android apk file description: http://developer.android.com/guide/components/fundamentals.html
49. Manifest files: http://developer.android.com/guide/topics/manifest/manifest-intro.html
50. Android Media Formats: http://developer.android.com/guide/appendix/media-formats.html
51. RTC Hardware Coding Requirements: http://www.webmproject.org/hardware/rtc-coding-requirements/
52. AudioEffect API: http://developer.android.com/reference/android/media/audiofx/AudioEffect.html
53. `Android android.content.pm.PackageManager` class and Hardware Features List:
http://developer.android.com/reference/android/content/pm/PackageManager.html
54. HTTP Live Streaming Draft Protocol: http://tools.ietf.org/html/draft-pantos-http-live-streaming-03
55. ADB: http://developer.android.com/tools/help/adb.html
56. Dumpsys: https://source.android.com/devices/tech/input/dumpsys.html
57. DDMS: http://developer.android.com/tools/debugging/ddms.html
58. Monkey testing tool: http://developer.android.com/tools/help/monkey.html
59. SysyTrace tool: http://developer.android.com/tools/help/systrace.html
60. Android Application Development-Related Settings:
http://developer.android.com/reference/android/provider/Settings.html#ACTION_APPLICATION_DEVELOPMENT_SETTINGS
61. Supporting Multiple Screens: http://developer.android.com/guide/practices/screens_support.html
62. `android.util.Display`Metrics: http://developer.android.com/reference/android/util/DisplayMetrics.html
63. RenderScript: http://developer.android.com/guide/topics/renderscript/
64. Android extension pack for OpenGL ES: https://developer.android.com/reference/android/opengl/GLES31Ext.html
65. Hardware Acceleration: http://developer.android.com/guide/topics/graphics/hardware-accel.html
66. EGL Extension-EGL_ANDROID_RECORDABLE:
http://www.khronos.org/registry/egl/extensions/ANDROID/EGL_ANDROID_recordable.txt
67. Display Manager: http://developer.android.com/reference/android/hardware/display/DisplayManager.html
68. `android.content.res.Configuration`:
http://developer.android.com/reference/android/content/res/Configuration.html

69. Action Assist: http://developer.android.com/reference/android/content/Intent.html#ACTION_ASSIST
70. Touch Input Configuration: http://source.android.com/devices/tech/input/touch-devices.html
71. Motion Event API: http://developer.android.com/reference/android/view/MotionEvent.html
72. Key Event API: http://developer.android.com/reference/android/view/KeyEvent.html
73. Android Open Source sensors: http://source.android.com/devices/sensors/
74. `android.hardware.SensorEvent`:
http://developer.android.com/reference/android/hardware/SensorEvent.html
75. Timestamp sensor event: http://developer.android.com/reference/android/hardware/SensorEvent.html#timestamp
76. Android Open Source composite sensors: http://source.android.com/devices/sensors/composite_sensors.html
77. Continuous trigger mode: http://source.android.com/devices/sensors/base_triggers.html#continuous
78. Accelerometer sensor:
http://developer.android.com/reference/android/hardware/Sensor.html#TYPE_ACCELEROMETER
79. Wi-Fi Multicast API: http://developer.android.com/reference/android/net/wifi/WifiManager.MulticastLock.html
80. Wi-Fi Direct (Wi-Fi P2P): http://developer.android.com/reference/android/net/wifi/p2p/WifiP2pManager.html
81. WifiManager API: http://developer.android.com/reference/android/net/wifi/WifiManager.html
82. Bluetooth API: http://developer.android.com/reference/android/bluetooth/package-summary.html
83. Bluetooth `ScanFilter` API: https://developer.android.com/reference/android/bluetooth/le/ScanFilter.html
84. NDEF Push Protocol: http://source.android.com/compatibility/ndef-push-protocol.pdf
85. Android Beam: http://developer.android.com/guide/topics/connectivity/nfc/nfc.html
86. Android NFC Sharing Settings:
http://developer.android.com/reference/android/provider/Settings.html#ACTION_NFCSHARING_SETTINGS
87. NFC Connection Handover: http://www.nfc-forum.org/specs/spec_list/#conn_handover
88. Bluetooth Secure Simple Pairing Using NFC:
http://members.nfc-forum.org/apps/group_public/download.php/18688/NFCForum-AD-BTSSP_1_1.pdf
89. Content Resolver: http://developer.android.com/reference/android/content/ContentResolver.html
90. Camera orientation API:
http://developer.android.com/reference/android/hardware/Camera.html#setDisplayOrientation(int)
91. Camera: http://developer.android.com/reference/android/hardware/Camera.html
92. Camera: http://developer.android.com/reference/android/hardware/Camera.Parameters.html
93. Camera hardware level:
https://developer.android.com/reference/android/hardware/camera2/CameraCharacteristics.html#INFO_SUPPORTED_HARDWARE_LEVEL
94. Camera version support: http://source.android.com/devices/camera/versioning.html
95. Android DownloadManager: http://developer.android.com/reference/android/app/DownloadManager.html
96. Android File Transfer: http://www.android.com/filetransfer
97. Android Open Accessories: http://developer.android.com/guide/topics/usb/accessory.html
98. Android USB Audio:
http://developer.android.com/reference/android/hardware/usb/UsbConstants.html#USB_CLASS_AUDIO
99. USB Charging Specification: http://www.usb.org/developers/docs/devclass_docs/USB_Battery_Charging_1.2.pdf
100. USB Host API: http://developer.android.com/guide/topics/usb/host.html

101. Wired audio headset: http://source.android.com/accessories/headset-spec.html
102. Android Security and Permissions reference: http://developer.android.com/guide/topics/security/permissions.html
103. UserManager reference: http://developer.android.com/reference/android/os/UserManager.html
104. External Storage reference: http://source.android.com/devices/tech/storage
105. External Storage APIs: http://developer.android.com/reference/android/os/Environment.html
106. SMS Short Code: http://en.wikipedia.org/wiki/Short_code
107. Android Open Source Encryption: http://source.android.com/devices/tech/encryption/index.html
108. Android Compatibility Program Overview: http://source.android.com/compatibility/index.html
109. Android Compatibility forum: https://groups.google.com/forum/#!forum/android-compatibility
110. WebM project: http://www.webmproject.org/

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.