android compatibility program

Android Compatibility Definition: Android 1.6

Android 1.6 r2
Google Inc.
compatibility@android.com

Table of Contents

1. Introduction

This document enumerates the requirements that must be met in order for mobile phones to be compatible with Android 1.6. This definition assumes familiarity with the Android Compatibility Program [Resources, 1].

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, 2].

As used in this document, a "device implementer" or "implementer" is a person or organization developing a hardware/software solution running Android 1.6. A "device implementation" or "implementation" is the hardware/software solution so developed.

To be considered compatible with Android 1.6, device implementations:

1. MUST meet the requirements presented in this Compatibility Definition, including any documents incorporated via reference.
2. MUST pass the Android Compatibility Test Suite (CTS) available as part of the Android Open Source Project [Resources, 3]. The CTS tests most, **but not all**, components outlined in this document.

Where this definition or the CTS 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, 4] is both the reference *and preferred* implementation of Android. Device implementers are strongly encouraged to base their implementations 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 CTS 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.

2. Resources

This Compatibility Definition makes reference to a number of resources that can be obtained here.

1. Android Compatibility Program Overview: https://sites.google.com/a/android.com/compatibility/how-it-works
2. IETF RFC2119 Requirement Levels: http://www.ietf.org/rfc/rfc2119.txt
3. Compatibility Test Suite: http://sites.google.com/a/android.com/compatibility/compatibility-test-suite--cts
4. Android Open Source Project: http://source.android.com/
5. API definitions and documentation: http://developer.android.com/reference/packages.html
6. Content Providers: http://code.google.com/android/reference/android/provider/package-summary.html
7. Available Resources: http://code.google.com/android/reference/available-resources.html
8. Android Manifest files: http://code.google.com/android/devel/bblocks-manifest.html
9. Android Permissions reference: http://developer.android.com/reference/android/Manifest.permission.html
10. Build Constants: http://developer.android.com/reference/android/os/Build.html
11. WebView: http://developer.android.com/reference/android/webkit/WebView.html
12. Gears Browser Extensions: http://code.google.com/apis/gears/

13. Dalvik Virtual Machine specification, found in the dalvik/docs directory of a source code checkout; also available at http://android.git.kernel.org/?p=platform/dalvik.git;a=tree;f=docs;h=3e2ddbcaf7f370246246f9f03620a7caccbfcb12;hb=HEAD
14. AppWidgets: http://developer.android.com/guide/practices/ui_guidelines/widget_design.html
15. Notifications: http://developer.android.com/guide/topics/ui/notifiers/notifications.html
16. Status Bar icon style guide: http://developer.android.com/guide/practices/ui_guideline/icon_design.html#statusbarstructure
17. Search Manager: http://developer.android.com/reference/android/app/SearchManager.html
18. Toast: http://developer.android.com/reference/android/widget/Toast.html
19. Apps For Android: http://code.google.com/p/apps-for-android
20. Android apk file description: http://developer.android.com/guide/topics/fundamentals.html
21. Android Debug Bridge (adb): http://code.google.com/android/reference/adb.html
22. Dalvik Debug Monitor Service (ddms): http://code.google.com/android/reference/ddms.html
23. Monkey: http://developer.android.com/guide/developing/tools/monkey.html
24. Display-Independence Documentation:
25. Configuration Constants: http://developer.android.com/reference/android/content/res/Configuration.html
26. Display Metrics: http://developer.android.com/reference/android/util/DisplayMetrics.html
27. Camera: http://developer.android.com/reference/android/hardware/Camera.html
28. Sensor coordinate space: http://developer.android.com/reference/android/hardware/SensorEvent.html
29. Android Security and Permissions reference: http://developer.android.com/guide/topics/security/security.html

Many of these resources are derived directly or indirectly from the Android 1.6 SDK, and will be functionally identical to the information in that SDK's documentation. In any cases where this Compatibility Definition 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.

3. Software

The Android platform includes both a set of managed ("hard") APIs, and a body of so-called "soft" APIs such as the Intent system, native-code APIs, and web-application APIs. This section details the hard and soft APIs that are integral to compatibility, as well as certain other relevant technical and user interface behaviors. Device implementations MUST comply with all the requirements in this section.

3.1. Managed API Compatibility

The managed (Dalvik-based) 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 VM environment. Device implementations MUST provide complete implementations, including all documented behaviors, of any documented API exposed by the Android 1.6 SDK, such as:

1. Core Android Java-language APIs [Resources, 5].
2. Content Providers [Resources, 6].
3. Resources [Resources, 7].
4. AndroidManifest.xml attributes and elements [Resources, 8].

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.

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 such as Intents, permissions, and similar aspects of Android applications that cannot be enforced at application compile time. This section details the "soft" APIs and system behaviors required for compatibility with Android 1.6. Device implementations MUST meet all the requirements presented in this section.

3.2.1. Permissions

Device implementers MUST support and enforce all permission constants as documented by the Permission reference page [Resources, 9]. Note that Section 10 lists addtional 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, 10] 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	Comments
android.os.Build.VERSION.RELEASE	The version of the currently-executing Android system, in human-readable format. For Android 1.6, this field MUST have the string value "1.6".
android.os.Build.VERSION.SDK	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 1.6, this field MUST have the integer value 4.
android.os.Build.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 re-used for different builds shipped 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 ("").
android.os.Build.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. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
android.os.Build.BRAND	A value chosen by the device implementer identifying the name of the company, organization, individual, etc. who produced the device, in human-readable format. A possible use of this field is to indicate the OEM

	and/or carrier who sold the device. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
android.os.Build.DEVICE	A value chosen by the device implementer identifying the specific configuration or revision of the body (sometimes called "industrial design") of the device. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
android.os.Build.FINGERPRINT	A string that uniquely identifies this build. It SHOULD be reasonably human-readable. It MUST follow this template: $(PRODUCT_BRAND)/$(PRODUCT_NAME)/$(PRODUCT_DEVICE)/ $(TARGET_BOOTLOADER_BOARD_NAME):$(PLATFORM_VERSION)/ $(BUILD_ID)/$(BUILD_NUMBER):$(TARGET_BUILD_VARIANT)/ $(BUILD_VERSION_TAGS) For example: acme/mydevicel/generic/generic:Donut/ERC77/ 3359:userdebug/test-keys The fingerprint MUST NOT include spaces. If other fields included in the template above have spaces, they SHOULD be replaced with the ASCII underscore ("_") character in the fingerprint.
android.os.Build.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 ("").
android.os.Build.ID	An identifier chosen by the device implementer to refer to a specific release, in human readable format. This field can by the same as android.os.Build.VERSION.INCREMENTAL, but SHOULD be a value intended to be somewhat meaningful for end users. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
android.os.Build.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 ("").
android.os.Build.PRODUCT	A value chosen by the device implementer containing the development name or code name of the device. MUST be human-readable, but is not necessarily intended for view by end users. There are no requirements on the specific format of this field, except that it MUST NOT be null or the empty string ("").
android.os.Build.TAGS	A comma-separated list of tags chosen by the device implementer that further distinguish the build. For example, "unsigned,debug". This field MUST NOT be null or the empty string (""), but a single tag (such as "release") is fine.
android.os.Build.TIME	A value representing the timestamp of when the build occurred.
android.os.Build.TYPE	A value chosen by the device implementer specifying the runtime configuration of the build. This field SHOULD have one of the values corresponding to the three typical Android runtime configurations: "user", "userdebug", or "eng".
android.os.Build.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

Android uses Intents to achieve loosely-coupled integration between applications. This section describes requirements related to the Intent patterns that MUST be honored by device implementations. By "honored", it is meant that the device implementer MUST provide an Android Activity, Service, or other component that specifies a matching Intent filter and binds to and implements correct behavior for each specified Intent pattern.

3.2.3.1. Core Application Intents

The Android upstream project defines a number of core applications, such as a phone dialer, calendar, contacts book, music player, and so on. Device implementers MAY replace these applications with alternative versions.

However, any such alternative versions MUST honor the same Intent patterns provided by the upstream project. (For example, if a device contains an alternative music player, it must still honor the Intent pattern issued by third-party applications to pick a song.) Device implementions MUST support all Intent patterns listed in Appendix A.

3.2.3.2. Intent Overrides

As Android is an extensible platform, device implementers MUST allow each Intent pattern described in Appendix A to be overridden by third-party applications. The upstream Android open source project 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 disabling the "Chooser" user interface which allows the user to select between multiple applications which all handle the same Intent pattern.

3.2.3.3. Intent Namespaces

Device implementers 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.* 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 listed in Appendices A or B.

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. A list of required Broadcast Intents is provided in Appendix B; however, note that the SDK may define additional broadcast intents, which MUST also be honored.

3.3. Native API Compatibility

Managed code running in Dalvik can call into native code provided in the application .apk file as an ELF .so file compiled for the appropriate device hardware architecture. Device implementations MUST include support for code running in the managed environment to call into native code, using the standard Java Native Interface (JNI) semantics. The following APIs must be available to native code:

- **libc (C library)**
- **libm (math library)**
- **JNI interface**
- **libz (Zlib compression)**
- **liblog (Android logging)**
- **Minimal support for C++**
- **OpenGL ES 1.1**

These libraries MUST be source-compatible (i.e. header compatible) and binary-compatible (for a given processor architecture) with the versions provided in Bionic by the Android Open Source project. Since the Bionic implementations are not fully compatible with other implementations such as the GNU C library, device implementers SHOULD use the Android implementation. If device implementers use a different implementation of these libraries, they must ensure header and binary compatibility.

Native code compatibility is challenging. For this reason, we wish to repeat that device implementers are VERY strongly encouraged to use the upstream implementations of the libraries listed above, to help ensure compatibility.

3.4. Web API Compatibility

Many developers and applications rely on the behavior of the android.webkit.WebView class [Resources, 11] for their user interfaces, so the WebView implementation must be compatible across Android implementations. The Android Open Source implementation uses the WebKit rendering engine version to implement the WebView.

Because it is not feasible to develop a comprehensive test suite for a web browser, device implementers MUST use the specific upstream build of WebKit in the WebView implementation. Specifically:

- WebView MUST use the 528.5+ WebKit build from the upstream Android Open Source tree for Android 1.6. This build includes a specific set of functionality and security fixes for the WebView.
- The user agent string reported by the WebView MUST be in this format:
  `Mozilla/5.0 (Linux; U; Android 1.6; <language>-<country>; <device name>; Build/<build ID>) AppleWebKit/528.5+ (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1`

- The "<device name>" string MUST be the same as the value for android.os.Build.MODEL
- The "<build ID>" string MUST be the same as the value for android.os.Build.ID.
- The "<language>" and "<country>" strings SHOULD follow the usual conventions for country code and language, and SHOULD refer to the curent locale of the device at the time of the request.

Implementations MAY ship a custom user agent string in the standalone Browser application. What's more, the standalone Browser MAY be based on an alternate browser technology (such as Firefox, Opera, etc.) However, even if an alternate Browser application is shipped, the WebView component provided to third-party applications MUST be based on WebKit, as above.

The standalone Browser application SHOULD include support for Gears [Resources, 12] and MAY include support for some or all of HTML5.

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 Android available from the Android Open Source Project.

Some specific areas of compatibility are:

- Devices MUST NOT change the behavior or meaning 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 particular permission

The above list is not comprehensive, and the onus is on device implementers to ensure behavioral compatibility. 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.

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.

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 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.

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. Virtual Machine Compatibility

A compatible Android device must support the full Dalvik Executable (DEX) bytecode specification and Dalvik Virtual Machine semantics [Resources, 13].

3.8. User Interface Compatibility

The Android platform includes some developer APIs that allow developers to hook into the system user interface. Device implementations MUST incorporate these standard UI APIs into custom user interfaces they develop, as explained below.

3.8.1. Widgets

Android defines a component type and corresponding API and lifecycle that allows applications to expose an "AppWidget" to the end user [Resources, 14]. The Android Open Source reference release includes a Launcher application that includes user interface elements allowing the user to add, view, and remove AppWidgets from the home screen.

Device implementers MAY substitute an alternative to the reference Launcher (i.e. home screen). Alternative Launchers SHOULD include built-in support for AppWidgets, and expose user interface elements to add, view, and remove AppWidgets directly within the Launcher. Alternative Launchers MAY omit these user interface elements; however, if they are omitted, the device implementer MUST provide a separate application accessible from the Launcher that allows users to add, view, and remove AppWidgets.

3.8.2. Notifications

Android includes APIs that allow developers to notify users of notable events [Resources, 15]. Device implementers MUST provide support for each class of notification so defined; specifically: sounds, vibration, light and status bar.

Additionally, the implementation MUST correctly render and all resources (icons, sound files, etc.) provided for in the APIs [Resources, 7], or in the Status Bar icon style guide [Resources, 16]. 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.

3.8.3. Search

Android includes APIs [Resources, 17] 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.

Device implementations MUST include a single, shared, system-wide search user interface capable of real-time suggestions in response to user input. Device implementations MUST implement the APIs that allow developers to reuse this user interface to provide search within their own applications.

Device implementations 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.

Device implementations MAY ship alternate search user interfaces, but SHOULD include a hard or soft dedicated search button, that can be used at any time within any app to invoke the search framework, with the behavior provided for in the API documentation.

3.8.4. Toasts

Applications can use the "Toast" API (defined in [Resources, 18]) to display short non-modal strings to the end user, that disappear after a brief period of time. Device implementations MUST display Toasts from applications to end users in some high-visibility manner.

4. Reference Software Compatibility

Device implementers MUST test implementation compatibility using the following open-source applications:

- Calculator (included in SDK)
- Lunar Lander (included in SDK)
- ApiDemos (included in SDK)
- The "Apps for Android" applications [Resources, 19]

Each app above MUST launch and behave correctly on the implementation, for the implementation to be

considered compatible.

5. 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, 20].

Devices implementations MUST NOT extend either the .apk, Android Manifest, or Dalvik bytecode formats in such a way that would prevent those files from installing and running correctly on other compatible devices. Device implementers SHOULD use the reference upstream implementation of Dalvik, and the reference implementation's package management system.

6. Multimedia Compatibility

A compatible Android device must support the following multimedia codecs. All of these codecs are provided as software implementations in the preferred Android implementation from the Android Open Source Project [Resources, 4].

Please note that neither Google nor the Open Handset Alliance make any representation that these codecs are unencumbered by 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.

Audio				
Name	**Encoder**	**Decoder**	**Details**	**Files Supported**
AAC LC/LTP		X	Mono/Stereo content in any combination of standard bit rates up to 160 kbps and sampling rates between 8 to 48kHz	3GPP (.3gp) and MPEG-4 (.mp4, .m4a) files. No support for raw AAC (.aac)
HE-AACv1 (AAC+)		X	Mono/Stereo content in any combination of standard bit rates up to 96 kbps and sampling rates between 8 to 48kHz	3GPP (.3gp) and MPEG-4 (.mp4, .m4a) files. No support for raw AAC (.aac)
HE-AACv2 (enhanced AAC+)		X	Mono/Stereo content in any combination of standard bit rates up to 96 kbps and sampling rates between 8 to 48kHz	3GPP (.3gp) and MPEG-4 (.mp4, .m4a) files. No support for raw AAC (.aac)
AMR-NB	X	X	4.75 to 12.2 kbps sampled @ 8kHz	3GPP (.3gp) files
AMR-WB		X	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz	-3GPP (.3gp) files
MP3		X	Mono/Stereo 8-320Kbps constant (CBR) or variable bit-rate (VBR)	MP3 (.mp3) files
MIDI		X	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF.	Type 0 and 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rtttl, .rtx), OTA (.ota),

			Support for ringtone formats RTTTL/RTX, OTA, and iMelody	and iMelody (.imy)
Ogg Vorbis		X		.ogg
PCM		X	8- and 16-bit linear PCM (rates up to limit of hardware)	WAVE

Image				
Name	**Encoder**	**Decoder**	**Details**	**Files Supported**
JPEG	X	X	base+progressive	
GIF		X		
PNG	X	X		
BMP		X		

Video				
Name	**Encoder**	**Decoder**	**Details**	**Files Supported**
H.263	X	X		3GPP (.3gp) files
H.264		X		3GPP (.3gp) and MPEG-4 (.mp4) files
MPEG4 SP		X		3GPP (.3gp) file

7. Developer Tool Compatibility

Device implemenations MUST support the Android Developer Tools provided in the Android SDK. Specifically, Android-compatible devices MUST be compatible with:

- **Android Debug Bridge or adb** [Resources, 21]
  Device implementations MUST support all `adb` functions as documented in the Android SDK. The device-side `adb` daemon SHOULD be inactive by default, but there MUST be a user-accessible mechanism to turn on the Android Debug Bridge.
- **Dalvik Debug Monitor Service or ddms** [Resources, 22]
  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, 23]
  Device implementations MUST include the Monkey framework, and make it available for applications to use.

8. Hardware Compatibility

Android is intended to support device implementers creating innovative form factors and configurations. At the same time Android developers expect certain hardware, sensors and APIs across all Android device. This section lists the hardware features that all Android 1.6 compatible devices must support. In Android 1.6, the majority of hardware features (such as WiFi, compass, and accelerometer) are required.

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 defined in the Android SDK documentation.

8.1. Display

Android 1.6 includes facilities that perform certain automatic scaling and transformation operations under some circumstances, to ensure that third-party applications run reasonably well on hardware configurations for which they were not necessarily explicitly designed [Resources, 24]. Devices MUST properly implement these behaviors, as detailed in this section.

8.1.1. Standard Display Configurations

This table lists the standard screen configurations considered compatible with Android:

Screen Type	Width (Pixels)	Height (Pixels)	Diagonal Length Range (inches)	Screen Size Group	Screen Density Group
QVGA	240	320	2.6 - 3.0	Small	Low
WQVGA	240	400	3.2 - 3.5	Normal	Low
FWQVGA	240	432	3.5 - 3.8	Normal	Low
HVGA	320	480	3.0 - 3.5	Normal	Medium
WVGA	480	800	3.3 - 4.0	Normal	High
FWVGA	480	854	3.5 - 4.0	Normal	High
WVGA	480	800	4.8 - 5.5	Large	Medium
FWVGA	480	854	5.0 - 5.8	Large	Medium

Device implementations corresponding to one of the standard configurations above MUST be configured to report the indicated screen size to applications via the android.content.res.Configuration [Resources, 25] class.

Some .apk packages have manifests that do not identify them as supporting a specific density range. When running such applications, the following constraints apply:

- Device implementations MUST interpret any resources that are present as defaulting to "medium" (known as "mdpi" in the SDK documentation.)
- When operating on a "low" density screen, device implementations MUST scale down medium/mdpi assets by a factor of 0.75.
- When operating on a "high" density screen, device implementations MUST scale up medium/mdpi assets by a factor of 1.5.
- Device implementations MUST NOT scale assets within a density range, and MUST scale assets by exactly these factors between density ranges.

8.1.2. Non-Standard Display Configurations

Display configurations that do not match one of the standard configurations listed in Section 8.2.1 require additional consideration and work to be compatible. Device implementers MUST contact Android Compatibility Team as provided for in Section 12 to obtain classifications for screen-size bucket, density, and scaling factor. When provided with this information, device implementations MUST implement them as specified.

Note that some display configurations (such as very large or very small screens, and some aspect ratios) are fundamentally incompatible with Android 1.6; therefore device implementers are encouraged to contact Android Compatibility Team as early as possible in the development process.

8.1.3. Display Metrics

Device implementations MUST report correct values for all display metrics defined in android.util.DisplayMetrics [Resources, 26].

8.2. Keyboard

Device implementations:

- MUST include support for the Input Management Framework (which allows third party developers to create Input Management Engines -- i.e. soft keyboard) as detailed at developer.android.com
- MUST provide at least one soft keyboard implementation (regardless of whether a hard keyboard is present)
- 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 [Resources, 25] (that is, QWERTY, or 12-key)

8.3. Non-touch Navigation

Device implementations:

- MAY omit non-touch navigation options (that is, may omit a trackball, 5-way directional pad, or wheel)
- MUST report via android.content.res.Configuration [Resources, 25] the correct value for the device's hardware

8.4. Screen Orientation

Compatible devices 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.

8.5. Touchscreen input

Device implementations:

- MUST have a touchscreen
- MAY have either capacative or resistive touchscreen
- MUST report the value of android.content.res.Configuration [Resources, 25] reflecting corresponding to the type of the specific touchscreen on the device

8.6. USB

Device implementations:

- MUST implement a USB client, connectable to a USB host with a standard USB-A port
- MUST implement the Android Debug Bridge over USB (as described in Section 7)
- MUST implement a USB mass storage client for the removable/media storage is present in the device
- SHOULD use the micro USB form factor on the device side
- SHOULD implement support for the USB Mass Storage specification (so that either removable or fixed storage on the device can be accessed from a host PC)
- MAY include a non-standard port on the device side, but if so MUST ship with a cable capable of connecting the custom pinout to standard USB-A port

8.7. Navigation keys

The Home, Menu and Back functions are essential to the Android navigation paradigm. Device implementations MUST make these functions available to the user at all times, regardless of application state. These functions SHOULD be implemented via dedicated buttons. They MAY be implemented using software, gestures, touch panel, etc., but if so they MUST be always accessible and not obscure or interfere with the available application display area.

Device implementers SHOULD also provide a dedicated search key. Device implementers MAY also provide send and end keys for phone calls.

8.8. WiFi

Device implementations MUST support 802.11b and 802.11g, and MAY support 802.11a.

8.9. Camera

Device implementations MUST include a camera. The included camera:

- 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.

Device implementations MUST implement the following behaviors for the camera-related APIs [Resources, 27]:

1. 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.
2. 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. (This is the format used natively by the 7k hardware family.) That is, NV21 MUST be the default.

8.9.1. Non-Autofocus Cameras

If a device lacks an autofocus camera, the device implementer MUST meet the additional requirements in this section. Device implementations MUST implement the full Camera API included in the Android 1.6 SDK documentation in some reasonable way, regardless of actual camera hardware's capabilities.

For Android 1.6, if the camera lacks auto-focus, the device implementation MUST adhere to the following:

1. The system MUST include a read-only system property named "ro.workaround.noautofocus" with the value of "1". This value is intended to be used by applications such as Android Market to selectively identify device capabilities, and will be replaced in a future version of Android with a robust API.
2. If an application calls android.hardware.Camera.autoFocus(), the system MUST call the onAutoFocus() callback method on any registered android.hardware.Camera.AutoFocusCallback instances, even though no focusing actually happened. This is to avoid having existing applications break by waiting forever for an autofocus callback that will never come.
3. The call to the AutoFocusCallback.onAutoFocus() method MUST be triggered by the driver or framework in a new event on the main framework Looper thread. That is, Camera.autoFocus() MUST NOT directly call AutoFocusCallback.onAutoFocus() since this violates the Android framework threading model and will break apps.

8.10. Accelerometer

Device implementations MUST include a 3-axis accelerometer and MUST be able to deliver events at at least 50 Hz. The coordinate system used by the accelerometer MUST comply with the Android sensor coordinate system as detailed in the Android APIs [Resources, 28].

8.11. Compass

Device implementations MUST include a 3-axis compass and MUST be able to deliver events at at least 10 Hz. The coordinate system used by the compass MUST comply with the Android sensor coordinate system as defined in the Android API [Resources, 28].

8.12. GPS

Device implementations MUST include a GPS, and SHOULD include some form of "assisted GPS" technique to minimize GPS lock-on time.

8.13. Telephony

Device implementations:

- MUST include either GSM or CDMA telephony
- MUST implement the appropriate APIs as detailed in the Android SDK documentation at developer.android.com

Note that this requirement implies that non-phone devices are not compatible with Android 1.6; Android 1.6 devices MUST include telephony hardware. Please see Appendix C for information on non-phone devices.

8.14. Volume controls

Android-compatible devices MUST include a mechanism to allow the user to increase and decrease the audio volume. Device implementations MUST make these functions available to the user at all times, regardless of application state. These functions MAY be implemented using physical hardware keys, software, gestures, touch panel, etc., but they MUST be always accessible and not obscure or interfere with the available application display area (see Display above).

When these buttons are used, the corresponding key events MUST be generated and sent to the foreground application. If the event is not intercepted and sunk by the application then device implementation MUST handle the event as a system volume control.

9. Performance Compatibility

One of the goals of the Android Compatibility Program is to ensure a consistent application experience for consumers. Compatible implementations must ensure not only that applications simply run correctly on the device, but that they do so with reasonable performance and overall good user experience.

Device implementations MUST meet the key performance metrics of an Android 1.6 compatible device, as in the table below:

Metric	Performance Threshold	Comments

Application Launch Time	The following applications should launch within the specified time. Browser: less than 1300ms MMS/SMS: less than 700ms AlarmClock: less than 650ms	This is tested by CTS. The launch time is measured as the total time to complete loading the default activity for the application, including the time it takes to start the Linux process, load the Android package into the Dalvik VM, and call onCreate.
Simultaneous Applications	Multiple applications will be launched. Re-launching the first application should complete taking less than the original launch time.	This is tested by CTS.

10. 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, 29] 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 following security mechanisms:

10.1. Permissions

Device implementations MUST support the Android permissions model as defined in the Android developer documentation [Resources, 9]. 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.

10.2. User and Process Isolation

Device implementations MUST support the Android application sandbox model, in which each application runs as a unique Unix-style 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, 29].

10.3. Filesystem Permissions

Device implementations MUST support the Android file access permissions model as defined in as defined in the Security and Permissions reference [Resources, 29].

11. Compatibility Test Suite

Device implementations MUST pass the Android Compatibility Test Suite (CTS) [Resources, 3] 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 1.6. However, such releases will only fix behavioral bugs in the CTS tests and will not impose any new tests, behaviors or APIs for a given platform release.

12. Contact Us

You can contact the Android Compatibility Team at compatibility@android.com for clarifications related to this Compatibiltiy Definition and to provide feedback on this Definition.

Appendix A: Required Application Intents

NOTE: this list is provisional, and will be updated in the future.

Application	Actions	Schemes	MIME Types
Browser	android.intent.action.VIEW	http https	(none) text/plain text/html application/xhtml+xml application/ vnd.wap.xhtml+xml
	android.intent.action.WEB_SEARCH	(none) http https	(none)
Camera	android.media.action.IMAGE_CAPTURE android.media.action.STILL_IMAGE_CAMERA android.media.action.VIDEO_CAMERA android.media.action.VIDEO_CAPTURE		
	android.intent.action.VIEW android.intent.action.GET_CONTENT android.intent.action.PICK android.intent.action.ATTACH_DATA		vnd.android.cursor.dir/ image vnd.android.cursor.dir/ video image/* video/*
	android.intent.action.VIEW	rtsp	
	android.intent.action.VIEW	http	video/mp4 video/3gp video/3gpp video/3gpp2
Phone / Contacts	android.intent.action.DIAL android.intent.action.VIEW android.intent.action.CALL	tel	
	android.intent.action.DIAL		
	android.intent.action.VIEW		vnd.android.cursor.dir/ person

	android.intent.action.PICK		vnd.android.cursor.dir/person vnd.android.cursor.dir/phone vnd.android.cursor.dir/postal-address
	android.intent.action.GET_CONTENT		vnd.android.cursor.item/person vnd.android.cursor.item/phone vnd.android.cursor.item/postal-address
Email	android.intent.action.SEND		text/plain image/* video/*
	android.intent.action.VIEW android.intent.action.SENDTO	mailto	
SMS / MMS	android.intent.action.VIEW android.intent.action.SENDTO	sms smsto mms mmsto	
Music	android.intent.action.VIEW	file	audio/* application/ogg application/x-ogg application/itunes
	android.intent.action.VIEW	http	audio/mp3 audio/x-mp3 audio/mpeg audio/mp4 audio/mp4a-latm
	android.intent.action.PICK		vnd.android.cursor.dir/artistalbum vnd.android.cursor.dir/album vnd.android.cursor.dir/nowplaying vnd.android.cursor.dir/track nd.android.cursor.dir/playlist vnd.android.cursor.dir/video
	android.intent.action.GET_CONTENT		media/* audio/* application/ogg application/x-ogg video/*

Package Installer	android.intent.action.VIEW	content file package	
	android.intent.action.PACKAGE_INSTALL	file http https	
	android.intent.action.ALL_APPS		
Settings	android.settings.SETTINGS android.settings.WIRELESS_SETTINGS android.settings.AIRPLANE_MODE_SETTINGS android.settings.WIFI_SETTINGS android.settings.APN_SETTINGS android.settings.BLUETOOTH_SETTINGS android.settings.DATE_SETTINGS android.settings.LOCALE_SETTINGS android.settings.INPUT_METHOD_SETTINGS com.android.settings.SOUND_SETTINGS com.android.settings.DISPLAY_SETTINGS android.settings.SECURITY_SETTING android.settings.LOCATION_SOURCE_SETTINGS android.settings.INTERNAL_STORAGE_SETTINGS android.settings.MEMORY_CARD_SETTINGS android.intent.action.SET_WALLPAPER		
Search	android.intent.action.SEARCH		query
	android.intent.action.SEARCH_LONG_PRESS		
Voice	android.intent.action.VOICE_COMMAND		

Contacts Management

Intent Action	Description
ATTACH_IMAGE	Starts an Activity that lets the user pick a contact to attach an image to.
EXTRA_CREATE_DESCRIPTION	Used with SHOW_OR_CREATE_CONTACT to specify an exact description to be

	shown when prompting user about creating a new contact.
EXTRA_FORCE_CREATE	**Used with SHOW_OR_CREATE_CONTACT to force creating a new contact if no matching contact found.**
SEARCH_SUGGESTION_CLICKED	**This is the intent that is fired when a search suggestion is clicked on.**
SEARCH_SUGGESTION_CREATE_CONTACT_CLICKED	**This is the intent that is fired when a search suggestion for creating a contact is clicked on.**
SEARCH_SUGGESTION_DIAL_NUMBER_CLICKED	**This is the intent that is fired when a search suggestion for dialing a number is clicked on.**
SHOW_OR_CREATE_CONTACT	**Takes as input a data URI with a mailto: or tel: scheme.**

Appendix B: Required Broadcast Intents **NOTE: this list is provisional, and will be updated in the future.**

Intent Action	Description
ACTION_BOOT_COMPLETED	Broadcast Action: This is broadcast once, after the system has finished booting.
ACTION_CALL_BUTTON	Broadcast Action: This is broadcast once, when a call is received.
ACTION_CAMERA_BUTTON	Broadcast Action: The "Camera Button" was pressed.
ACTION_CONFIGURATION_CHANGED	Broadcast Action: The current device Configuration (orientation, locale, etc) has changed.
ACTION_DATE_CHANGED	Broadcast Action: The date has changed.
ACTION_DEVICE_STORAGE_LOW	Broadcast Action: Indicates low memory condition on the device
ACTION_DEVICE_STORAGE_OK	Broadcast Action: Indicates low memory condition on the device no longer exists
ACTION_HEADSET_PLUG	Broadcast Action: Wired Headset plugged in or unplugged.
ACTION_INPUT_METHOD_CHANGED	Broadcast Action: An input method has been changed.
ACTION_MEDIA_BAD_REMOVAL	Broadcast Action: External media was removed from SD card slot, but mount point was not unmounted.
ACTION_MEDIA_BUTTON	Broadcast Action: The "Media Button" was pressed.
ACTION_MEDIA_CHECKING	Broadcast Action: External media is present, and being disk-checked The path to the mount point for the checking media is contained in the Intent.mData field.
ACTION_MEDIA_EJECT	Broadcast Action: User has expressed the desire to remove the external storage media.
ACTION_MEDIA_MOUNTED	Broadcast Action: External media is present and mounted at its mount point.
ACTION_MEDIA_NOFS	Broadcast Action: External media is present, but is using an incompatible fs (or is blank) The path to the mount point for the checking media is contained in the Intent.mData field.
ACTION_MEDIA_REMOVED	Broadcast Action: External media has been removed.
ACTION_MEDIA_SCANNER_FINISHED	Broadcast Action: The media scanner has finished scanning a directory.
ACTION_MEDIA_SCANNER_SCAN_FILE	Broadcast Action: Request the media scanner to scan a file and add it to the media database.

ACTION_MEDIA_SCANNER_STARTED	Broadcast Action: The media scanner has started scanning a directory.
ACTION_MEDIA_SHARED	Broadcast Action: External media is unmounted because it is being shared via USB mass storage.
ACTION_MEDIA_UNMOUNTABLE	Broadcast Action: External media is present but cannot be mounted.
ACTION_MEDIA_UNMOUNTED	Broadcast Action: External media is present, but not mounted at its mount point.
ACTION_NEW_OUTGOING_CALL	Broadcast Action: An outgoing call is about to be placed.
ACTION_PACKAGE_ADDED	Broadcast Action: A new application package has been installed on the device.
ACTION_PACKAGE_CHANGED	Broadcast Action: An existing application package has been changed (e.g. a component has been enabled or disabled.
ACTION_PACKAGE_DATA_CLEARED	Broadcast Action: The user has cleared the data of a package. This should be preceded by ACTION_PACKAGE_RESTARTED, after which all of its persistent data is erased and this broadcast sent. Note that the cleared package does *not* receive this broadcast. The data contains the name of the package.
ACTION_PACKAGE_REMOVED	Broadcast Action: An existing application package has been removed from the device. The data contains the name of the package. The package that is being installed does *not* receive this Intent.
ACTION_PACKAGE_REPLACED	Broadcast Action: A new version of an application package has been installed, replacing an existing version that was previously installed.
ACTION_PACKAGE_RESTARTED	Broadcast Action: The user has restarted a package, and all of its processes have been killed. All runtime state associated with it (processes, alarms, notifications, etc) should be removed. Note that the restarted package does *not* receive this broadcast. The data contains the name of the package.
ACTION_PROVIDER_CHANGED	Broadcast Action: Some content providers have parts of their namespace where they publish new events or items that the user may be especially interested in.
ACTION_SCREEN_OFF	Broadcast Action: Sent after the screen turns off.
ACTION_SCREEN_ON	Broadcast Action: Sent after the screen turns on.
ACTION_UID_REMOVED	Broadcast Action: A user ID has been removed from the system.
ACTION_UMS_CONNECTED	Broadcast Action: The device has entered USB Mass Storage mode.

ACTION_UMS_DISCONNECTED	Broadcast Action: The device has exited USB Mass Storage mode.
ACTION_USER_PRESENT	Broadcast Action: Sent when the user is present after device wakes up (e.g when the keyguard is gone).
ACTION_WALLPAPER_CHANGED	Broadcast Action: The current system wallpaper has changed.
ACTION_TIME_CHANGED	Broadcast Action: The time was set.
ACTION_TIME_TICK	Broadcast Action: The current time has changed.
ACTION_TIMEZONE_CHANGED	Broadcast Action: The timezone has changed.
ACTION_BATTERY_CHANGED	Broadcast Action: The charging state, or charge level of the battery has changed.
ACTION_BATTERY_LOW	Broadcast Action: Indicates low battery condition on the device. This broadcast corresponds to the "Low battery warning" system dialog.
ACTION_BATTERY_OKAY	Broadcast Action: Indicates the battery is now okay after being low. This will be sent after ACTION_BATTERY_LOW once the battery has gone back up to an okay state.

Network State

Intent Action	Description
NETWORK_STATE_CHANGED_ACTION	Broadcast intent action indicating that the state of Wi-Fi connectivity has changed.
RSSI_CHANGED_ACTION	Broadcast intent action indicating that the RSSI (signal strength) has changed.
SUPPLICANT_STATE_CHANGED_ACTION	Broadcast intent action indicating that a connection to the supplicant has been established or lost.
WIFI_STATE_CHANGED_ACTION	Broadcast intent action indicating that Wi-Fi has been enabled, disabled, enabling, disabling, or unknown.
NETWORK_IDS_CHANGED_ACTION	The network IDs of the configured networks could have changed.
ACTION_BACKGROUND_DATA_SETTING_CHANGED	Broadcast intent action indicating that the setting for background data usage has changed values.
CONNECTIVITY_ACTION	Broadcast intent indicating that a change in network connectivity has occurred.
ACTION_AIRPLANE_MODE_CHANGED	Broadcast Action: The user has switched the phone into or out of Airplane Mode.

Appendix C: Future Considerations

This appendix clarifies certain portions of this Android 1.6 Compatibility Definition, and in some cases discusses anticipated or planned changes intended for a future version of the Android platform. This appendix is for informational and planning purposes only, and is not part of the Compatibility Definition for Android 1.6.

1. Non-telephone Devices

Android 1.6 is intended exclusively for telephones; telephony functionality is not optional. Future versions of the Android platform are expected to make telephony optional (and thus allow for non-phone Android devices), but only phones are compatible with Android 1.6.

2. Bluetooth Compatibility

The Android 1.6 release of Android does not support Bluetooth APIs, so from a compatibility perspective Bluetooth does not impose any considerations for this version of the platform. However, a future version of Android will introduce Bluetooth APIs. At that point, supporting Bluetooth will become mandatory for compatibility.

Consequently, we strongly recommend that Android 1.6 devices include Bluetooth, so that they will be compatible with future versions of Android that require Bluetooth.

3. Required Hardware Components

All hardware components in Section 8 (including WiFi, magnetometer/compass, accelerometer, etc.) are required and may not be omitted. Future versions of Android are expected to make some (but not all) of these components optional, in tandem with corresponding tools for third-party developers to handle these changes.

4. Sample Applications

The Compatibility Definition Document for a future version of Android will include a more extensive and representative list of applications than the ones listed in Section 4, above. For Android 1.6, the applications listed in Section 4 must be tested.

5. Touch Screens

Future versions of the Compatibility Definition may or may not allow for devices to omit touchscreens. However, currently much of the Android framework implementation assumes the existence of a touchscreen; omitting a touchscreen would break substantially all current third-party Android applications, so in Android 1.6 a touchscreen is required for compatibility.

6. Performance

Future versions of CTS will also measure the CPU utilization and performance of the following components of an implementation:

- 2D graphics
- 3D graphics
- Video playback
- Audio playback
- Bluetooth A2DP playback