android compatibility program

Android 2.3 Compatibility Definition

compatibility@android.com

Table of Contents

1. Introduction
2. Resources
3. Software
    3.1. Managed API Compatibility
    3.2. Soft API Compatibility
        3.2.1. Permissions
        3.2.2. Build Parameters
        3.2.3. Intent Compatibility
            3.2.3.1. Core Application Intents
            3.2.3.2. Intent Overrides
            3.2.3.3. Intent Namespaces
            3.2.3.4. Broadcast Intents
    3.3. Native API Compatibility
    3.4. Web Compatibility
        3.4.1. WebView Compatibility
        3.4.2. Browser Compatibility
    3.5. API Behavioral Compatibility
    3.6. API Namespaces
    3.7. Virtual Machine Compatibility
    3.8. User Interface Compatibility
        3.8.1. Widgets
        3.8.2. Notifications
        3.8.3. Search
        3.8.4. Toasts
        3.8.5. Live Wallpapers
4. Application Packaging Compatibility
5. Multimedia Compatibility
    5.1. Media Codecs
    5.1.1. Media Decoders
    5.1.2. Media Encoders
    5.2. Audio Recording
    5.3. Audio Latency
6. Developer Tool Compatibility
7. Hardware Compatibility
    7.1. Display and Graphics
        7.1.1. Screen Configurations
        7.1.2. Display Metrics
        7.1.3. Declared Screen Support
        7.1.4. Screen Orientation
        7.1.5. 3D Graphics Accleration
    7.2. Input Devices

7.2.1. Keyboard
7.2.2. Non-touch Navigation
7.2.3. Navigation keys
7.2.4. Touchscreen input
7.3. Sensors
7.3.1. Accelerometer
7.3.2. Magnetometer
7.3.3. GPS
7.3.4. Gyroscope
7.3.5. Barometer
7.3.6. Thermometer
7.3.7. Photometer
7.3.8. Proximity Sensor
7.4. Data Connectivity
7.4.1. Telephony
7.4.2. IEEE 802.11 (WiFi)
7.4.3. Bluetooth
7.4.4. Near-Field Communications
7.4.5. Minimum Network Capability
7.5. Cameras
7.5.1. Rear-Facing Camera
7.5.2. Front-Facing Camera
7.5.3. Camera API Behavior
7.5.4. Camera Orientation
7.6. Memory and Storage
7.6.1. Minimum Memory and Storage
7.6.2. Application Shared Storage
7.7. USB
8. Performance Compatibility
9. Security Model Compatibility
9.1. Permissions
9.2. UID and Process Isolation
9.3. Filesystem Permissions
9.4. Alternate Execution Environments
10. Software Compatibility Testing
10.1. Compatibility Test Suite
10.2. CTS Verifier
10.3. Reference Applications
11. Updatable Software
12. Contact Us
Appendix A - Bluetooth Test Procedure

1. Introduction

This document enumerates the requirements that must be met in order for mobile phones to be compatible with Android 2.3.

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

To be considered compatible with Android 2.3, 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, 3] 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.

Please note that this Compatibility Definition is issued to correspond with the 2.3.3 update to Android, which is API level 10. This Definition obsoletes and replaces the Compatibility Definition for Android 2.3 versions prior to 2.3.3. (That is, versions 2.3.1 and 2.3.2 are obsolete.) Future Android-compatible devices running Android 2.3 MUST ship with version 2.3.3 or later.

2. Resources

1. IETF RFC2119 Requirement Levels: http://www.ietf.org/rfc/rfc2119.txt
2. Android Compatibility Program Overview: http://source.android.com/compatibility/index.html
3. Android Open Source Project: http://source.android.com/
4. API definitions and documentation: http://developer.android.com/reference/packages.html
5. Android Permissions reference: http://developer.android.com/reference/android/Manifest.permission.html
6. android.os.Build reference: http://developer.android.com/reference/android/os/Build.html
7. Android 2.3 allowed version strings: http://source.android.com/compatibility/2.3/versions.html
8. android.webkit.WebView class: http://developer.android.com/reference/android/webkit/WebView.html
9. HTML5: http://www.whatwg.org/specs/web-apps/current-work/multipage/
10. HTML5 offline capabilities: http://dev.w3.org/html5/spec/Overview.html#offline
11. HTML5 video tag: http://dev.w3.org/html5/spec/Overview.html#video
12. HTML5/W3C geolocation API: http://www.w3.org/TR/geolocation-API/
13. HTML5/W3C webdatabase API: http://www.w3.org/TR/webdatabase/
14. HTML5/W3C IndexedDB API: http://www.w3.org/TR/IndexedDB/
15. Dalvik Virtual Machine specification: available in the Android source code, at dalvik/docs
16. AppWidgets: http://developer.android.com/guide/practices/ui_guidelines/widget_design.html
17. Notifications: http://developer.android.com/guide/topics/ui/notifiers/notifications.html
18. Application Resources: http://code.google.com/android/reference/available-resources.html
19. Status Bar icon style guide: http://developer.android.com/guide/practices/ui_guideline /icon_design.html#statusbarstructure
20. Search Manager: http://developer.android.com/reference/android/app/SearchManager.html
21. Toasts: http://developer.android.com/reference/android/widget/Toast.html
22. Live Wallpapers: http://developer.android.com/resources/articles/live-wallpapers.html
23. Reference tool documentation (for adb, aapt, ddms): http://developer.android.com/guide/developing/tools/index.html
24. Android apk file description: http://developer.android.com/guide/topics/fundamentals.html
25. Manifest files: http://developer.android.com/guide/topics/manifest/manifest-intro.html

26. Monkey testing tool: http://developer.android.com/guide/developing/tools/monkey.html
27. Android Hardware Features List: http://developer.android.com/reference/android/content/pm/PackageManager.html
28. Supporting Multiple Screens: http://developer.android.com/guide/practices/screens_support.html
29. android.util.DisplayMetrics: http://developer.android.com/reference/android/util/DisplayMetrics.html
30. android.content.res.Configuration: http://developer.android.com/reference/android/content/res/Configuration.html
31. Sensor coordinate space: http://developer.android.com/reference/android/hardware/SensorEvent.html
32. Bluetooth API: http://developer.android.com/reference/android/bluetooth/package-summary.html
33. NDEF Push Protocol: http://source.android.com/compatibility/ndef-push-protocol.pdf
34. MIFARE MF1S503X: http://www.nxp.com/documents/data_sheet/MF1S503x.pdf
35. MIFARE MF1S703X: http://www.nxp.com/documents/data_sheet/MF1S703x.pdf
36. MIFARE MF0ICU1: http://www.nxp.com/documents/data_sheet/MF0ICU1.pdf
37. MIFARE MF0ICU2: http://www.nxp.com/documents/short_data_sheet/MF0ICU2_SDS.pdf
38. MIFARE AN130511: http://www.nxp.com/documents/application_note/AN130511.pdf
39. MIFARE AN130411: http://www.nxp.com/documents/application_note/AN130411.pdf
40. Camera orientation API: http://developer.android.com/reference/android/hardware/Camera.html#setDisplayOrientation(int)
41. android.hardware.Camera: http://developer.android.com/reference/android/hardware/Camera.html
42. Android Security and Permissions reference: http://developer.android.com/guide/topics/security/security.html
43. Apps for Android: http://code.google.com/p/apps-for-android

Many of these resources are derived directly or indirectly from the Android 2.3 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.

3. Software

The Android platform includes a set of managed APIs, a set of native APIs, and a body of so-called "soft" APIs such as the Intent system 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 2.3 SDK [Resources, 4].

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 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 2.3. 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, 5]. Note that Section 10 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, 6] 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. This field MUST have one of the string values defined in [Resources, 7].
android.os.Build.VERSION.SDK	The version of the currently-executing Android system, in a format accessible to third-party application code. For Android 2.3, this field MUST have the integer value 9.
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 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 ("").
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. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "`^[a-zA-Z0-9.,_-]+$`".
android.os.Build.BRAND	A value chosen by the device implementer identifying the name of the company, organization, individual, etc. who produced the device, in human-readable format. A possible use of this field is to indicate the OEM and/or carrier who sold the device. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "`^[a-zA-Z0-9.,_-]+$`".
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. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "`^[a-zA-Z0-9.,_-]+$`".

android.os.Build.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/mydevice/generic/generic:2.3/GRC77/3359: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.

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 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.,_-]+$`".
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. The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "^[a-zA-Z0-9.,_-]+$".
android.os.Build.TAGS	A comma-separated list of tags chosen by the device implementer that further distinguish the build. For example, "unsigned,debug". The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "^[a-zA-Z0-9.,_-]+$".
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". The value of this field MUST be encodable as 7-bit ASCII and match the regular expression "^[a-zA-Z0-9.,_-]+$".
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 or Service 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.

The following applications are considered core Android system applications:

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

The core Android system applications include various Activity, or Service components that are considered "public". That is, the attribute "android:exported" may be absent, or may have the value "true".

For every Activity or Service defined in one of the core Android system apps that is not marked as non-public via an android:exported attribute with the value "false", device implementations MUST include a compontent of the same type implementing the same Intent filter patterns as the core Android system app.

In other words, a device implementation MAY replace core Android system apps; however, if it does, the device implementation MUST support all Intent patterns defined by each core Android system app being replaced.

3.2.3.2. Intent Overrides

As Android is an extensible platform, device implementers MUST allow each Intent pattern referenced in Section 3.2.3.1 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 but is not limited to 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 used by the core apps listed in Section 3.2.3.1.

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.

3.3. 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. As native code is highly dependent on the underlying processor technology, Android defines a number of Application Binary Interfaces (ABIs) in the Android NDK, in the file `docs/CPU-ARCH-ABIS.txt`. If a device implementation is compatible with one or more defined ABIs, it SHOULD 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 accurately report the native Application Binary Interface (ABI) supported by the device, via the `android.os.Build.CPU_ABI` API
- MUST report only those ABIs documented in the latest version of the Android NDK, in the file `docs/CPU-ARCH-ABIS.txt`
- 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 1.0)
- libGLESv2.so (OpenGL ES 2.0)
- libEGL.so (native OpenGL surface management)
- libjnigraphics.so
- libOpenSLES.so (Open Sound Library audio support)
- libandroid.so (native Android activity 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 ABI at all.

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

3.4. Web Compatibility

Many developers and applications rely on the behavior of the `android.webkit.WebView` class [Resources, 8] for their user interfaces, so the WebView implementation must be compatible across Android implementations. Similarly, a complete, modern web broswer is central to the Android user experience. Device implementations MUST include a version of `android.webkit.WebView` consistent with the upstream Android software, and MUST include a modern HTML5-capable browser, as described below.

3.4.1. WebView Compatibility

The Android Open Source implementation uses the WebKit rendering engine to implement the `android.webkit.WebView`. 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 WebKit in the WebView implementation. Specifically:

- Device implementations' `android.webkit.WebView` implementations MUST be based on the 533.1 WebKit build from the upstream Android Open Source tree for Android 2.3. This build includes a specific set of functionality and security fixes for the WebView. Device implementers MAY include customizations to the WebKit implementation; however, any such customizations MUST NOT alter the behavior of the WebView, including rendering behavior.

  The user agent string reported by the WebView MUST be in this format:
  `Mozilla/5.0 (Linux; U; Android $(VERSION); $(LOCALE); $(MODEL) Build/$(BUILD)) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1`
  - The value of the $(VERSION) string MUST be the same as the value for `android.os.Build.VERSION.RELEASE`
  - The value of the $(LOCALE) string SHOULD follow the ISO conventions for country code and language, and SHOULD refer to the current configured locale of the device
  - 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 WebView component SHOULD include support for as much of HTML5 [Resources, 9] as possible. Minimally, device implementations MUST support each of these APIs associated with HTML5 in the WebView:

- application cache/offline operation [Resources, 10]
- the <video> tag [Resources, 11]
- geolocation [Resources, 12]

Additionally, device implementations MUST support the HTML5/W3C webstorage API [Resources, 13], and SHOULD support the HTML5/W3C IndexedDB API [Resources, 14]. *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.*

HTML5 APIs, like all JavaScript APIs, MUST be disabled by default in a WebView, unless the developer explicitly enables them via the usual Android APIs.

3.4.2. Browser Compatibility

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, 9] as possible. Minimally, device implementations MUST support each of these APIs associated with HTML5:

- application cache/offline operation [Resources, 10]
- the <video> tag [Resources, 11]
- geolocation [Resources, 12]

Additionally, device implementations MUST support the HTML5/W3C webstorage API [Resources, 13], and SHOULD support the HTML5/W3C IndexedDB API [Resources, 14]. *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, 3]. 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. Virtual Machine Compatibility

Device implementations MUST support the full Dalvik Executable (DEX) bytecode specification and Dalvik Virtual Machine semantics [Resources, 15].

Device implementations with screens classified as medium- or low-density MUST configure Dalvik to allocate at least 16MB of memory to each application. Device implementations with screens classified as high-density or extra-high-density MUST configure Dalvik to allocate at least 24MB of memory to each application. Note that device implementations MAY allocate more memory than these figures.

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, 16]. 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, configure, 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, configure, view, and remove AppWidgets.

3.8.2. Notifications

Android includes APIs that allow developers to notify users of notable events [Resources, 17]. 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 all resources (icons, sound files, etc.) provided for in the APIs [Resources, 18], or in the Status Bar icon style guide [Resources, 19]. 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, 20] 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, 21]) 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.

3.8.5. 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, 22]. 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 framerate with no adverse affects 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 Open GL 1.0 or 2.0 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. Device implementations determined to not run live wallpapers reliably as described above MUST NOT implement live wallpapers.

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, 23].

Devices implementations MUST NOT extend either the .apk [Resources, 24], Android Manifest [Resources, 25], or Dalvik bytecode [Resources, 15] 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.

5. Multimedia Compatibility

Device implementations MUST fully implement all multimedia APIs. Device implementations MUST include support for all multimedia codecs described below, and SHOULD meet the sound processing guidelines described below. Device implementations MUST include at least one form of audio output, such as speakers, headphone jack, external speaker connection, etc.

5.1. Media Codecs

Device implementations MUST support the multimedia codecs as detailed in the following sections. 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 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.

The tables below do not list specific bitrate requirements for most video codecs. The reason for this is that in practice, current device hardware does not necessarily support bitrates that map exactly to the required bitrates specified by the relevant standards. Instead, device implementations SHOULD support the highest bitrate practical on the hardware, up to the limits defined by the specifications.

5.1.1. Media Decoders

Device implementations MUST include an implementation of an decoder for each codec and format described in the table below. Note that decoders for each of these media types are provided by the upstream Android Open-Source Project.

	Name	Details	File/Container Format
Audio	AAC LC/LTP	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). No support for raw AAC (.aac)
	HE-AACv1 (AAC+)		
	HE-AACv2 (enhanced AAC+)		
	AMR-NB	4.75 to 12.2 kbps sampled @ 8kHz	3GPP (.3gp)
	AMR-WB	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz	3GPP (.3gp)
	MP3	Mono/Stereo 8-320Kbps constant (CBR) or variable bit-rate (VBR)	MP3 (.mp3)
	MIDI	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	Type 0 and 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rttl, .rtx), OTA (.ota), and iMelody (.imy)
	Ogg Vorbis		Ogg (.ogg)
	PCM	8- and 16-bit linear PCM (rates up to limit of hardware)	WAVE (.wav)
Image	JPEG	base+progressive	
	GIF		
	PNG		
	BMP		
Video	H.263		3GPP (.3gp) files
	H.264		3GPP (.3gp) and MPEG-4 (.mp4) files
	MPEG4 Simple Profile		3GPP (.3gp) file

5.1.2. Media Encoders

Device implementations SHOULD include encoders for as many of the media formats listed in Section 5.1.1. as possible. However, some encoders do not make sense for devices that lack certain optional hardware; for instance, an encoder for the H.263 video does not make sense, if the device lacks any cameras. Device implementations MUST therefore implement media encoders according to the conditions described in the table below.

See Section 7 for details on the conditions under which hardware may be omitted by device implementations.

	Name	Details	File/Container Format	Conditions
Audio	AMR-NB	4.75 to 12.2 kbps sampled @ 8kHz	3GPP (.3gp)	Device implementations that include microphone hardware and define `android.hardware.microphone` MUST include encoders for these audio formats.
	AMR-WB	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz	3GPP (.3gp)	
	AAC LC/LTP	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).	

Image	JPEG	base+progressive		All device implementations MUST include encoders for these image formats, as Android 2.3 includes APIs that applications can use to programmatically generate files of these types.
	PNG			
Video	H.263		3GPP (.3gp) files	Device implementations that include camera hardware and define either `android.hardware.camera` or `android.hardware.camera.front` MUST include encoders for these video formats.

In addition to the encoders listed above, device implementations SHOULD include an H.264 encoder. Note that the Compatibility Definition for a future version is planned to change this requirement to "MUST". That is, H.264 encoding is optional in Android 2.3 but **will be required** by a future version. Existing and new devices that run Android 2.3 are **very strongly encouraged to meet this requirement in Android 2.3**, or they will not be able to attain Android compatibility when upgraded to the future version.

5.2. Audio Recording

When an application has used the `android.media.AudioRecord` API to start recording an audio stream, device implementations SHOULD sample and record audio with each of these behaviors:

- Noise reduction processing, if present, SHOULD be disabled.
- Automatic gain control, if present, SHOULD be disabled.
- 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 5000 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% from 100 Hz to 4000 Hz at 90 dB SPL input level.

Note: while the requirements outlined above are stated as "SHOULD" for Android 2.3, the Compatibility Definition for a future version is planned to change these to "MUST". That is, these requirements are optional in Android 2.3 but **will be required** by a future version. Existing and new devices that run Android 2.3 are **very strongly encouraged to meet these requirements in Android 2.3**, or they will not be able to attain Android compatibility when upgraded to the future version.

5.3. Audio Latency

Audio latency is broadly defined as the interval between when an application requests an audio playback or record operation, and when the device implementation actually begins the operation. Many classes of applications rely on short latencies, to achieve real-time effects such sound effects or VOIP communication. Device implementations that include microphone hardware and declare `android.hardware.microphone` SHOULD meet all audio latency requirements outlined in this section. See Section 7 for details on the conditions under which microphone hardware may be omitted by device implementations.

For the purposes of this section:

- "cold output latency" is defined to be the interval between when an application requests audio playback and when sound begins playing, when the audio system has been idle and powered down prior to the request
- "warm output latency" is defined to be the interval between when an application requests audio playback and when sound begins playing, when the audio system has been recently used but is currently idle (that is, silent)
- "continuous output latency" is defined to be the interval between when an application issues a sample to be played and when the speaker physically plays the corresponding sound, while the device is currently playing back audio
- "cold input latency" is defined to be the interval between when an application requests audio recording and when the first sample is delivered to the application via its callback, when the audio system and microphone has been idle and powered down prior to the request

- "continuous input latency" is defined to be when an ambient sound occurs and when the sample corresponding to that sound is delivered to a recording application via its callback, while the device is in recording mode

Using the above definitions, device implementations SHOULD exhibit each of these properties:

- cold output latency of 100 milliseconds or less
- warm output latency of 10 milliseconds or less
- continuous output latency of 45 milliseconds or less
- cold input latency of 100 milliseconds or less
- continuous input latency of 50 milliseconds or less

Note: while the requirements outlined above are stated as "SHOULD" for Android 2.3, the Compatibility Definition for a future version is planned to change these to "MUST". That is, these requirements are optional in Android 2.3 but **will be required** by a future version. Existing and new devices that run Android 2.3 are **very strongly encouraged to meet these requirements in Android 2.3**, or they will not be able to attain Android compatibility when upgraded to the future version.

If a device implementation meets the requirements of this section, 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, 27] Conversely, if the device implementation does not meet these requirements it MUST NOT report support for low-latency audio.

6. Developer Tool Compatibility

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

- **Android Debug Bridge (known as adb)** [Resources, 23]
  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 (known as ddms)** [Resources, 23]
  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, 26]
  Device implementations MUST include the Monkey framework, and make it available for applications to use.

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, and Windows 7, in both 32-bit and 64-bit versions.

7. Hardware Compatibility

Android is intended to enable device implementers to create innovative form factors and configurations. At the same time Android developers write innovative applications that rely on the various hardware and features available through the Android APIs. The requirements in this section strike a balance between innovations available to device implementers, and the needs of developers to ensure their apps are only available to devices where they will run properly.

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 present
- 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 accurately report accurate hardware configuration information via the `getSystemAvailableFeatures()` and `hasSystemFeature(String)` methods on the `android.content.pm.PackageManager` class. [Resources, 27]

7.1. Display and Graphics

Android 2.3 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, 28]. Devices MUST properly implement these APIs and behaviors, as detailed in this section.

7.1.1. Screen Configurations

Device implementations MAY use screens of any pixel dimensions, provided that they meet the following requirements:

- screens MUST be at least 2.5 inches in physical diagonal size
- density MUST be at least 100 dpi
- the aspect ratio MUST be between 1.333 (4:3) and 1.779 (16:9)
- the display technology used consists of square pixels

Device implementations with a screen meeting the requirements above are considered compatible, and no additional action is necessary. The Android framework implementation automatically computes display characteristics such as screen size bucket and density bucket. In the majority of cases, the framework decisions are the correct ones. If the default framework computations are used, no additional action is necessary. Device implementers wishing to change the defaults, or use a screen that does not meet the requirements above MUST contact the Android Compatibility Team for guidance, as provided for in Section 12.

The units used by the requirements above are defined as follows:

- "Physical diagonal size" is the distance in inches between two opposing corners of the illuminated portion of the display.
- "dpi" (meaning "dots per inch") is 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" is 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".

Device implementations MUST use only displays with a single static configuration. That is, device implementations MUST NOT enable multiple screen configurations. For instance, since a typical television supports multiple resolutions such as 1080p, 720p, and so on, this configuration is not compatible with Android 2.3. (However, support for such configurations is under investigation and planned for a future version of Android.)

7.1.2. Display Metrics

Device implementations MUST report correct values for all display metrics defined in `android.util.DisplayMetrics` [Resources, 29].

7.1.3. Declared Screen Support

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, medium, and large screens, as described in the Android SDK documentation.

7.1.4. 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 that cannot be physically rotated MAY meet this requirement by "letterboxing" applications that request portrait mode, using only a portion of the available display.

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.

7.1.5. 3D Graphics Acceleration

Device implementations MUST support OpenGL ES 1.0, as required by the Android 2.3 APIs. For devices that lack 3D acceleration hardware, a software implementation of OpenGL ES 1.0 is provided by the upstream Android Open-Source Project. Device implementations SHOULD support OpenGL ES 2.0.

Implementations MAY omit Open GL ES 2.0 support; however if support is ommitted, device implementations MUST NOT report as supporting OpenGL ES 2.0. Specifically, if a device implementations lacks OpenGL ES 2.0 support:

- the managed APIs (such as via the `GLES10.getString()` method) MUST NOT report support for OpenGL ES 2.0
- the native C/C++ OpenGL APIs (that is, those available to apps via libGLES_v1CM.so, libGLES_v2.so, or libEGL.so) MUST NOT report support for OpenGL ES 2.0.

Conversely, if a device implementation *does* support OpenGL ES 2.0, it MUST accurately report that support via the routes just listed.

Note that Android 2.3 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 2.3 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.

7.2. Input Devices

Android 2.3 supports a number of modalities for user input. Device implementations MUST support user input devices as provided for in this section.

7.2.1. 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.keyboard` [Resources, 30] (that is, QWERTY, or 12-key)

7.2.2. Non-touch Navigation

Device implementations:

- MAY omit a non-touch navigation option (that is, may omit a trackball, d-pad, or wheel)
- MUST report the correct value for `android.content.res.Configuration.navigation` [Resources, 30]
- 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 code includes a selection mechanism suitable for use with devices that lack non-touch navigation inputs.

7.2.3. 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.

7.2.4. Touchscreen input

Device implementations:

- MUST have a touchscreen
- MAY have either capacitive or resistive touchscreen
- MUST report the value of `android.content.res.Configuration` [Resources, 30] reflecting corresponding to the type of the specific touchscreen on the device
- SHOULD support fully independently tracked pointers, if the touchscreen supports multiple pointers

7.3. Sensors

Android 2.3 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. For example, device implementations:

- MUST accurately report the presence or absence of sensors per the `android.content.pm.PackageManager` class. [Resources, 27]
- 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.)

The list above is not comprehensive; the documented behavior of the Android SDK is to be considered authoritative.

Some sensor types are synthetic, 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.

The Android 2.3 APIs introduce a notion of a "streaming" sensor, which is one that returns data continuously, rather than only when the data changes. Device implementations MUST continuously provide periodic data samples for any API indicated by the Android 2.3 SDK documentation to be a streaming sensor.

7.3.1. Accelerometer

Device implementations SHOULD include a 3-axis accelerometer. If a device implementation does include a 3-axis accelerometer, it:

- MUST be able to deliver events at 50 Hz or greater
- MUST comply with the Android sensor coordinate system as detailed in the Android APIs (see [Resources, 31])
- MUST be capable of measuring from freefall up to twice gravity (2g) or more on any three-dimensional vector
- MUST have 8-bits of accuracy or more
- MUST have a standard deviation no greater than 0.05 m/s^2

7.3.2. Magnetometer

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

- MUST be able to deliver events at 10 Hz or greater
- MUST comply with the Android sensor coordinate system as detailed in the Android APIs (see [Resources, 31]).
- MUST be capable of sampling a range of field strengths adequate to cover the geomagnetic field
- MUST have 8-bits of accuracy or more
- MUST have a standard deviation no greater than 0.5 µT

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 (i.e. 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 be capable of measuring orientation changes up to 5.5*Pi radians/second (that is, approximately 1,000 degrees per second)

- MUST be able to deliver events at 100 Hz or greater
- MUST have 8-bits of accuracy or more

7.3.5. Barometer

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

- MUST be able to deliver events at 5 Hz or greater
- MUST have adequate precision to enable estimating altitude

7.3.7. Thermometer

Device implementations MAY but SHOULD NOT include a thermometer (i.e. temperature sensor.) If a device implementation does include a thermometer, it MUST measure the temperature of the device CPU. It MUST NOT measure any other temperature. (Note that this sensor type is deprecated in the Android 2.3 APIs.)

7.3.7. Photometer

Device implementations MAY include a photometer (i.e. ambient light sensor.)

7.3.8. Proximity Sensor

Device implementations MAY 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. If a device implementation has a proximity sensor, it MUST be have 1-bit of accuracy or more.

7.4. Data Connectivity

Network connectivity and access to the Internet are vital features of Android. Meanwhile, device-to-device interaction adds significant value to Android devices and applications. Device implementations MUST meet the data connectivity requirements in this section.

7.4.1. Telephony

"Telephony" as used by the Android 2.3 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 2.3 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 sub-features, regardless of whether they use a cellular network for data connectivity.

Android 2.3 MAY be used on devices that do not include telephony hardware. That is, Android 2.3 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 (WiFi)

Android 2.3 device implementations SHOULD include support for one or more forms of 802.11 (b/g/a/n, etc.) If a device implementation does include support for 802.11, it MUST implement the corresponding Android API.

7.4.3. Bluetooth

Device implementations SHOULD include a Bluetooth transceiver. Device implementations that do include a Bluetooth transceiver MUST enable the RFCOMM-based Bluetooth API as described in the SDK documentation [Resources, 32]. Device implementations SHOULD implement relevant Bluetooth profiles, such as A2DP, AVRCP, OBEX, etc. as appropriate for the device.

The Compatibility Test Suite includes cases that cover basic operation of the Android RFCOMM Bluetooth API. However, since Bluetooth is a communications protocol between devices, it cannot be fully tested by unit tests running on a single device. Consequently, device implementations MUST also pass the human-driven Bluetooth test procedure described in Appendix A.

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, then it:

- MUST report the android.hardware.nfc feature from the `android.content.pm.PackageManager.hasSystemFeature()` method. [Resources, 27]
- 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)
    - NfcV (ISO 15693)
    - IsoDep (ISO 14443-4)
    - NFC Forum Tag Types 1, 2, 3, 4 (defined by the NFC Forum)
  - 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, 33]
  - MUST scan for all supported technologies while in NFC discovery mode.
  - SHOULD be in NFC discovery mode while the device is awake with the screen active.

(Note that publicly available links are not available for the JIS, ISO, and NFC Forum specifications cited above.)

Additionally, device implementations SHOULD support the following widely-deployed MIFARE technologies.

- MIFARE Classic (NXP MF1S503x [Resources, 34], MF1S703x [Resources, 35])
- MIFARE Ultralight (NXP MF0ICU1 [Resources, 36], MF0ICU2 [Resources, 37])
- NDEF on MIFARE Classic (NXP AN130511 [Resources, 38], AN130411 [Resources, 39])

Note that Android 2.3.3 includes APIs for these MIFARE types. If a device implementation supports MIFARE, 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, 27] 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, 27], and MUST implement the Android 2.3 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, 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 (WiFi).

Devices MAY implement more than one form of data connectivity.

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.

7.5.1. Rear-Facing Camera

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

- 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 a front-facing camera, it:

- MUST have a resolution of at least VGA (that is, 640x480 pixels)
- MUST NOT use a front-facing camera as the default for the Camera API. That is, the camera API in Android 2.3 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, 40] 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 data returned to any "postview" camera callback handlers, in the same manner as the camera preview image stream. (If the device implementation does not support postview callbacks, 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. Camera API Behavior

Device implementations MUST implement the following behaviors for the camera-related APIs, for both front- and rear-facing cameras:

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. That is, NV21 MUST be the default.
3. Device implementations SHOULD support the YV12 format (as denoted by the `android.graphics.ImageFormat.YV12` constant) for camera previews for both front- and rear-facing cameras. Note that the Compatibility Definition for a future version is planned to change this requirement to "MUST". That is, YV12 support is optional in Android 2.3 but **will be required** by a future version. Existing and new devices that run Android 2.3 are **very strongly encouraged to meet this requirement in Android 2.3**, or they will not be able to attain Android compatibility when upgraded to the future version.

Device implementations MUST implement the full Camera API included in the Android 2.3 SDK documentation [Resources, 41]), 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.

7.5.4. 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 dimention. That is, when the device is held in the landscape orientation, a 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

The fundamental function of Android 2.3 is to run applications. Device implementations MUST the requirements of this section, to ensure adequate storage and memory for applications to run properly.

7.6.1. Minimum Memory and Storage

Device implementations MUST have at least 128MB of memory available to the kernel and userspace. The 128MB MUST be in addition to any memory dedicated to hardware components such as radio, memory, and so on that is not under the kernel's control.

Device implementations MUST have at least 150MB of non-volatile storage available for user data. That is, the `/data` partition MUST be at least 150MB.

Beyond the requirements above, device implementations SHOULD have at least 1GB of non-volatile storage available for user data. Note that this higher requirement is planned to become a hard minimum in a future version of Android. Device implementations are strongly encouraged to meet these requirements now, or else they may not be eligible for compatibility for a future version of Android.

The Android APIs include a Download Manager that applications may use to download data files. The Download Manager implementation MUST be capable of downloading individual files 55MB in size, or larger. The Download Manager implementation SHOULD be capable of downloading files 100MB in size, or larger.

7.6.2. Application Shared Storage

Device implementations MUST offer shared storage for applications. The shared storage provided MUST be at least 1GB in size.

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 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 MAY have hardware for user-accessible removable storage, such as a Secure Digital card. Alternatively, device implementations MAY allocate internal (non-removable) storage as shared storage for apps.

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 or Media Transfer Protocol.

It is illustrative to consider two common examples. If a device implementation includes an SD card slot to satisfy the shared storage requirement, a FAT-formatted SD card 1GB in size or larger MUST be included with the device as sold to users, and MUST be mounted by default. Alternatively, if a device implementation uses internal fixed storage to satisfy this 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 that include multiple shared storage paths (such as both an SD card slot and shared internal storage) SHOULD modify the core applications such as the media scanner and ContentProvider to transparently support files placed in both locations.

7.7. 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 the USB mass storage specification, to allow a host connected to the device to access the contents of the /sdcard volume
- SHOULD use the micro USB form factor on the device side
- 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. Performance Compatibility

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 2.3 compatible device defined 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	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	When multiple applications have been launched, re-launching an already-running application after it has been launched must take less than the original launch time.	

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, 42] 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 sub-sections.

9.1. Permissions

Device implementations MUST support the Android permissions model as defined in the Android developer documentation [Resources, 42]. 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 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, 42].

9.3. Filesystem Permissions

Device implementations MUST support the Android file access permissions model as defined in as defined in the Security and Permissions reference [Resources, 42].

9.4. Alternate Execution Environments

Device implementations MAY include runtime environments that execute applications using some other software or technology than the Dalvik virtual machine 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 (that is, Linux user IDs, etc.)
- Alternate runtimes MAY provide a single Android sandbox shared by all applications using the alternate runtime.
- Alternate runtimes 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
- Alternate runtimes MUST NOT launch with, grant, or be granted access to the sandboxes corresponding to other Android applications.

Alternate runtimes MUST NOT be launched with, be granted, or grant to other applications any privileges of the superuser (root), or of any other user ID.

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. That is, 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.

10. Software Compatibility Testing

The Android Open-Source Project includes various testing tools to verify that device implementations are compatible. 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 2.3 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, 2] 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 2.3. Device implementations MUST pass the latest CTS version available at the time the device software is completed.

MUST pass the most recent version of the Android Compatibility Test Suite (CTS) available at the time of the device implementation's software is completed. (The CTS is available as part of the Android Open Source Project [Resources, 2].) The CTS tests many, but not all, of the components outlined in this document.

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 which 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 Verfier only by the set of included locales, branding, etc. MAY omit the CTS Verifier test.

10.3. Reference Applications

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

- The "Apps for Android" applications [Resources, 43].
- Replica Island (available in Android Market; only required for device implementations that support with OpenGL ES 2.0)

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

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

The update mechanism used MUST support updates without wiping user data. Note that the upstream Android software includes an update mechanism that 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. Contact Us

You can contact the document authors at compatibility@android.com for clarifications and to bring up any issues that you think the document does not cover.

Appendix A - Bluetooth Test Procedure

The Compatibility Test Suite includes cases that cover basic operation of the Android RFCOMM Bluetooth API. However, since Bluetooth is a communications protocol between devices, it cannot be fully tested by unit tests running on a single device. Consequently, device implementations MUST also pass the human-operated Bluetooth test procedure described below.

The test procedure is based on the BluetoothChat sample app included in the Android open-source project tree. The procedure requires two devices:

- a candidate device implementation running the software build to be tested
- a separate device implementation already known to be compatible, and of a model from the device implementation being tested -- that is, a "known good" device implementation

The test procedure below refers to these devices as the "candidate" and "known good" devices, respectively.

Setup and Installation

1. Build BluetoothChat.apk via 'make samples' from an Android source code tree.
2. Install BluetoothChat.apk on the known-good device.
3. Install BluetoothChat.apk on the candidate device.

Test Bluetooth Control by Apps

1. Launch BluetoothChat on the candidate device, while Bluetooth is disabled.
2. Verify that the candidate device either turns on Bluetooth, or prompts the user with a dialog to turn on Bluetooth.

Test Pairing and Communication

1. Launch the Bluetooth Chat app on both devices.
2. Make the known-good device discoverable from within BluetoothChat (using the Menu).
3. On the candidate device, scan for Bluetooth devices from within BluetoothChat (using the Menu) and pair with the known-good device.
4. Send 10 or more messages from each device, and verify that the other device receives them correctly.
5. Close the BluetoothChat app on both devices by pressing **Home**.
6. Unpair each device from the other, using the device Settings app.

Test Pairing and Communication in the Reverse Direction

1. Launch the Bluetooth Chat app on both devices.
2. Make the candidate device discoverable from within BluetoothChat (using the Menu).
3. On the known-good device, scan for Bluetooth devices from within BluetoothChat (using the Menu) and pair with the candidate device.
4. Send 10 or messages from each device, and verify that the other device receives them correctly.
5. Close the Bluetooth Chat app on both devices by pressing Back repeatedly to get to the Launcher.

Test Re-Launches

1. Re-launch the Bluetooth Chat app on both devices.
2. Send 10 or messages from each device, and verify that the other device receives them correctly.

Note: the above tests have some cases which end a test section by using Home, and some using Back. These tests are not redundant and are not optional: the objective is to verify that the Bluetooth API and stack works correctly both when Activities are explicitly terminated (via the user pressing Back, which calls finish()), and implicitly sent to background (via the user pressing Home.) Each test sequence MUST be performed as

described.