Porting, Extending, and Customizing

Embedded Android

O'REILLY®

Karim Yaghmour

Embedded Android

by Karim Yaghmour

Copyright © 2013 Karim Yaghmour. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Andy Oram and Mike Hendrickson
Production Editor: Kara Ebrahim
Copyeditor: Rebecca Freed
Proofreader: Julie Van Keuren
Indexer: Bob Pfahler
Cover Designer: Randy Comer
Interior Designer: David Futato
Illustrator: Rebecca Demarest

March 2013: First Edition

Revision History for the First Edition:

2013-03-11: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449308292* for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Embedded Android*, the image of a Moorish wall gecko, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-30829-2

[LSI]

Table of Contents

Preface .... **xi**

1. Introduction **1**
History 1
Features and Characteristics 2
Development Model 5
Differences From "Classic" Open Source Projects 5
Feature Inclusion, Roadmaps, and New Releases 7
Ecosystem 7
A Word on the Open Handset Alliance 8
Getting "Android" 9
Legal Framework 10
Code Licenses 10
Branding Use 13
Google's Own Android Apps 15
Alternative App Markets 15
Oracle versus Google 15
Mobile Patent Warfare 16
Hardware and Compliance Requirements 17
Compliance Definition Document 18
Compliance Test Suite 21
Development Setup and Tools 22

2. Internals Primer **25**
App Developer's View 25
Android Concepts 26
Framework Intro 30
App Development Tools 31
Native Development 32

Overall Architecture 33
Linux Kernel 34
Wakelocks 35
Low-Memory Killer 37
Binder 39
Anonymous Shared Memory (ashmem) 40
Alarm 41
Logger 42
Other Notable Androidisms 45
Hardware Support 46
The Linux Approach 46
Android's General Approach 47
Loading and Interfacing Methods 49
Device Support Details 51
Native User-Space 52
Filesystem Layout 53
Libraries 54
Init 57
Toolbox 58
Daemons 59
Command-Line Utilities 60
Dalvik and Android's Java 60
Java Native Interface (JNI) 63
System Services 63
Service Manager and Binder Interaction 68
Calling on Services 70
A Service Example: the Activity Manager 70
Stock AOSP Packages 71
System Startup 73

3. AOSP Jump-Start. 79
Development Host Setup 79
Getting the AOSP 80
Inside the AOSP 86
Build Basics 91
Build System Setup 91
Building Android 94
Running Android 99
Using the Android Debug Bridge (ADB) 101
Mastering the Emulator 105

4. The Build System. 111

Comparison with Other Build Systems 111
Architecture 113
Configuration 115
envsetup.sh 118
Function Definitions 124
Main Make Recipes 125
Cleaning 127
Module Build Templates 128
Output 132
Build Recipes 134
The Default droid Build 134
Seeing the Build Commands 134
Building the SDK for Linux and Mac OS 135
Building the SDK for Windows 136
Building the CTS 136
Building the NDK 137
Updating the API 138
Building a Single Module 139
Building Out of Tree 140
Building Recursively, In-Tree 142
Basic AOSP Hacks 143
Adding a Device 143
Adding an App 148
Adding an App Overlay 149
Adding a Native Tool or Daemon 150
Adding a Native Library 151

5. Hardware Primer. 155
Typical System Architecture 155
The Baseband Processor 157
Core Components 158
Real-World Interaction 159
Connectivity 160
Expansion, Development, and Debugging 160
What's in a System-on-Chip (SoC)? 161
Memory Layout and Mapping 165
Development Setup 169
Evaluation Boards 171

6. Native User-Space. 175
Filesystem 175
The Root Directory 179

/system 180
/data 182
SD Card 185
The Build System and the Filesystem 185
adb 191
Theory of Operation 191
Main Flags, Parameters, and Environment Variables 193
Basic Local Commands 194
Device Connection and Status 195
Basic Remote Commands 197
Filesystem Commands 202
State-Altering Commands 204
Tunneling PPP 207
Android's Command Line 208
The Shell Up to 2.3/Gingerbread 209
The Shell Since 4.0/Ice-Cream Sandwich 210
Toolbox 211
Core Native Utilities and Daemons 220
Extra Native Utilities and Daemons 227
Framework Utilities and Daemons 228
Init 228
Theory of Operation 228
Configuration Files 230
Global Properties 238
ueventd 243
Boot Logo 245

7. Android Framework. 249
Kick-Starting the Framework 250
Core Building Blocks 250
System Services 254
Boot Animation 257
Dex Optimization 260
Apps Startup 262
Utilities and Commands 266
General-Purpose Utilities 266
Service-Specific Utilities 278
Dalvik Utilities 292
Support Daemons 297
installd 298
vold 299
netd 301

rild 302
keystore 303
Other Support Daemons 304
Hardware Abstraction Layer 304

A. Legacy User-Space. 307

B. Adding Support for New Hardware. 323

C. Customizing the Default Lists of Packages. 337

D. Default init.rc Files. 341

E. Resources. 367

Index. 373

CHAPTER 1

Introduction

Putting Android on an embedded device is a complex task involving an intricate understanding of its internals and a clever mix of modifications to the Android Open Source Project (AOSP) and the kernel on which it runs, Linux. Before we get into the details of embedding Android, however, let's start by covering some essential background that embedded developers should factor in when dealing with Android, such as Android's hardware requirements, as well as the legal framework surrounding Android and its implications within an embedded setting. First, let's look at where Android comes from and how it was developed.

History

The story goes[1] that back in early 2002, Google's Larry Page and Sergey Brin attended a talk at Stanford about the development of the then-new Sidekick phone by Danger Inc. The speaker was Andy Rubin, Danger's CEO at the time, and the Sidekick was one of the first multifunction, Internet-enabled devices. After the talk, Larry went up to look at the device and was happy to see that Google was the default search engine. Soon after, both Larry and Sergey became Sidekick users.

Despite its novelty and enthusiastic users, however, the Sidekick didn't achieve commercial success. By 2003, Rubin and Danger's board agreed it was time for him to leave. After trying out a few things, Rubin decided he wanted to get back into the phone OS business. Using a domain name he owned, *android.com*, he set out to create an open OS for phone manufacturers. After investing most of his savings in the project and

1. Coinciding with Android's initial announcement in November 2007, *The New York Times* ran an article entitled "I, Robot: The Man Behind the Google Phone" by John Markoff, which gave an insightful background portrait of Andy Rubin and his career. By extension, it provided a lot of insight on the story behind Android. This section is partly based on that article.

having received some additional seed money, he set out to get the company funded. Soon after, in August 2005, Google acquired Android Inc. with little fanfare.

Between its acquisition and its announcement to the world in November 2007, Google released little to no information about Android. Instead, the development team worked furiously on the OS while deals and prototypes were being worked on behind the scenes. The initial announcement was made by the Open Handset Alliance (OHA), a group of companies unveiled for the occasion with its stated mission being the development of open standards for mobile devices and Android being its first product. A year later, in September 2008, the first open source version of Android, 1.0, was made available.

Several Android versions have been released since then, and the OS's progression and development is obviously more public. As we will see later, though, much of the work on Android continues to be done behind closed doors. Table 1-1 provides a summary of the various Android releases and the most notable features found in the corresponding AOSP.

Table 1-1. Android versions

Version	Release date	Codename	Most notable feature(s)	Open source
1.0	September 2008	*Unknown*		Yes
1.1	February 2009	*Unknown*[a]		Yes
1.5	April 2009	Cupcake	Onscreen soft keyboard	Yes
1.6	September 2009	Donut	Battery usage screen and VPN support	Yes
2.0, 2.0.1, 2.1	October 2009	Eclair	Exchange support	Yes
2.2	May 2010	Froyo	Just-in-Time (JIT) compile	Yes
2.3	December 2010	Gingerbread	SIP and NFC support	Yes
3.0	January 2011	Honeycomb	Tablet form-factor support	No
3.1	May 2011	Honeycomb	USB host support and APIs	No
4.0	November 2011	Ice-Cream Sandwich	Merged phone and tablet form-factor support	Yes
4.1	June 2012	Jelly Bean	Lots of performance optimizations	Yes
4.2	November 2012	Jelly Bean	Multiuser support	Yes

[a] This version is rumored to have been called "Petit Four." Have a look at this Google+ post for more information.

Features and Characteristics

Around the time 2.3.x/Gingerbread was released, Google used to advertise the following features about Android on its developer site:

Application framework

The application framework used by app developers to create what is commonly referred to as Android apps. The use of this framework is documented online and in books like O'Reilly's *Learning Android*.

Dalvik virtual machine

The clean-room byte-code interpreter implementation used in Android as a replacement for the Sun Java virtual machine (VM). While the latter interprets *.class* files, Dalvik interprets *.dex* files. These files are generated by the *dx* utility using the *.class* files generated by the Java compiler part of the JDK.

Integrated browser

Android includes a WebKit-based browser as part of its standard list of applications. App developers can use the `WebView` class to use the WebKit engine within their own apps.

Optimized graphics

Android provides its own 2D graphics library but relies on OpenGL ES[2] for its 3D capabilities.

SQLite

This is the standard SQLite database found here and made available to app developers through the application framework.

Media support

Android provides support for a wide range of media formats through StageFright, its custom media framework. Prior to 2.2, Android used to rely on PacketVideo's OpenCore framework.

GSM telephony support[3]

The telephony support is hardware dependent, and device manufacturers must provide a HAL module to enable Android to interface with their hardware. HAL modules will be discussed in the next chapter.

Bluetooth, EDGE, 3G, and WiFi

Android includes support for most wireless connection technologies. While some are implemented in Android-specific fashion, such as EDGE and 3G, others are provided in the same way as in plain Linux, as in the case of Bluetooth and WiFi.

2. OpenGL *ES* is a version of the OpenGL standard aimed at embedded systems.

3. Android obviously supports more than just GSM telephony. Nevertheless, this is the feature's name as it was officially advertised.

Camera, GPS, compass, and accelerometer
: Interfacing with the user's environment is key to Android. APIs are made available in the application framework to access these devices, and some HAL modules are required to enable their support.

Rich development environment
: This is likely one of Android's greatest assets. The development environment available to developers makes it very easy to get started with Android. A full SDK is freely available to download, along with an emulator, an Eclipse plug-in, and a number of debugging and profiling tools.

There are of course a lot more features that could be listed for Android, such as USB support, multitasking, multitouch, SIP, tethering, voice-activated commands, etc., but the previous list should give you a good idea of what you'll find in Android. Also note that every new Android release brings in its own new set of features. Check the Platform Highlights published with every version for more information on features and enhancements.

In addition to its basic feature set, the Android platform has a few characteristics that make it an especially interesting platform for embedded development. Here's a quick summary:

Broad app ecosystem
: At the time of this writing, there were 700,000 apps in Google Play, previously known as the Android Market. This compares quite favorably to the Apple App Store's 700,000 apps and ensures that you have a large pool to choose from should you want to prepackage applications with your embedded device. Bear in mind that you likely need to enter into some kind of agreement with an app's publisher before you can package that app. The app's availability in Google Play doesn't imply the right for you as a third party to redistribute it.

Consistent app APIs
: All APIs provided in the application framework are meant to be forward-compatible. Hence, custom apps that you develop for inclusion in your embedded system should continue working in future Android versions. In contrast, modifications you make to Android's source code are not guaranteed to continue applying or even working in the next Android release.

Replaceable components
: Because Android is open source, and as a benefit of its architecture, a lot of its components can be replaced outright. For instance, if you don't like the default Launcher app (home screen), you can write your own. More fundamental changes

can also be made to Android. The GStreamer[4] developers, for example, were able to replace StageFright, the default media framework in Android, with GStreamer without modifying the app API.

Extendable
: Another benefit of Android's openness and its architecture is that adding support for additional features and hardware is relatively straightforward. You just need to emulate what the platform is doing for other hardware or features of the same type. For instance, you can add support for custom hardware to the HAL by adding a handful of files, as is explained in Appendix B.

Customizable
: If you'd rather use existing components, such as the existing Launcher app, you can still customize them to your liking. Whether it be tuning their behavior or changing their look and feel, you are again free to modify the AOSP as needed.

Development Model

When considering whether to use Android, it's crucial that you understand the ramifications its development process may have on any modifications you make to it or to any dependencies you may have on its internals.

Differences From "Classic" Open Source Projects

Android's open source nature is one of its most trumpeted features. Indeed, as we've just seen, many of the software engineering benefits that derive from being open source apply to Android.

Despite its licensing, however, Android is unlike most open source projects in that its development is done mostly behind closed doors. The vast majority of open source projects, for example, have public mailing lists and forums where the main developers can be found interacting with one another, and public source repositories providing access to the main development branch's tip. No such thing can be found for Android.

This is best summarized by Andy Rubin himself: "Open source is different than a community-driven project. Android is light on community-driven, somewhat heavy on open source."

Whether we like it or not, Android is mainly developed within Google by the Android development team, and the public is not privy to either internal discussions nor the tip of the development branch. Instead, Google makes code-drops every time a new version of Android ships on a new device, which is usually every six months. For instance, a

4. GStreamer is the default media framework used in most desktop Linux environments, including Gnome, KDE, and XFCE.

few days after the Samsung Nexus S was released in December 2010, the code for the new version of the Android it was running, 2.3/Gingerbread, was made publicly available at *http://android.googlesource.com/*.

Obviously there is a certain amount of discomfort in the open source community with the continued use of the term "open source" in the context of a project whose development model contradicts the standard modus operandi of open source projects, especially given Android's popularity. The open source community has not historically been well served by projects that have adopted a similar development model. Others fear this development model also makes them vulnerable to potential changes in Google's business objectives.

Political issues aside, though, Android's development model means that as a developer, your ability to make contributions to Android is limited. Indeed, unless you become part of the Android development team at Google, you will not be able to make contributions to the tip of the development branch. Also, save for a handful of exceptions, it's unlikely you will be able to discuss your enhancements one-on-one with the core development team members. However, you are still free to submit enhancements and fixes to the AOSP code dumps made available at *http://android.googlesource.com/*.

The worst side effect of Google's approach is that you have absolutely no way to get inside information about the platform decisions being made by the Android development team. If new features are added within the AOSP, for example, or if modifications are made to core components, you will find out how such changes are made and how they impact changes you might have made to a previous version only by analyzing the next code dump. Furthermore, you will have no way to learn about the underlying requirement, restriction, or issue that justified the modification or inclusion. Had this been a true open source project, a public mailing list archive would exist where all this information, or pointers to it, would be available.

That being said, it's important to remember how significant a contribution Google is making by distributing Android under an open source license. Despite its awkward development model from an open source community perspective, it remains that Google's work on Android is a godsend for a large number of developers. Plus, it has accomplished one thing no other open source project was ever able to: created a massively successful Linux distribution. It would, therefore, be hard to fault Android's development team for its work.

Furthermore, it can easily be argued that from a business and go-to-market perspective that a community-driven process would definitely knock the wind out of any product announcements Google would attempt to release, making it impossible to create "buzz" around press announcements and the like, since every new feature would be developed in the open. That is to say nothing of the nondeterministic nature of community-driven processes that can see a group of people take years to agree on the best way to implement a given feature set. And, simply based on track record, Android's success has definitely

benefited from Google's ability to rapidly move it forward and to generate press interest based on releases of cool new products.

Feature Inclusion, Roadmaps, and New Releases

In brief, there is no publicly available roadmap for features and capabilities in future Android releases. At best, Google will announce ahead of time the name and approximate release date of the next version. Usually you can expect a new Android release to be made in time for the Google I/O conference, which is typically held in May, and another release by year-end. What will be in that release, though, is anyone's guess.

Typically, however, Google will choose a single manufacturer to work with on the next Android release. During that period, Google will work very closely with that single manufacturer's engineers to ready the next Android version to work on a targeted upcoming lead (or flagship) device. During that period, the manufacturer's team is reported to have access to the tip of the development branch. Once the device is put on the market, the corresponding source code dump is made to the public repositories. For the next release, it chooses another manufacturer and starts over.

There is one notable exception to that cycle: Android 3.x/Honeycomb. In that specific case, Google didn't release the source code to the corresponding lead device, the Motorola Xoom. The rationale seems to have been that the Android development team essentially forked the Android codebase at some point in time to start getting a tablet-ready version of Android out ASAP, in response to market timing prerogatives. Hence, in that version, very little regard was given to preserving backward compatibility with the phone form factor. And given that, Google did not wish to make the code available to avoid fragmentation of its platform. Instead, both phone and tablet form factor support were merged into the subsequent Android 4.0/Ice-Cream Sandwich release.

Ecosystem

As of January 2013:

- 1.3 million Android phones are activated each day, up from 400,000 in June 2011 and 200,000 in August 2010.
- Google Play contains around 700,000 apps. In comparison, the Apple App Store has about the same number of apps.[5]
- Android holds 72% of the global smartphone market.

5. At the time of this writing, it's the first time ever that Google Play catches up to the number of apps in the App Store.

Android is clearly on the upswing. In fact, Gartner predicted in October 2012 that Android would be the dominant OS, besting the venerable Windows, by 2016. Much as Linux disrupted the embedded market about a decade ago, Android is poised to make its mark. Not only will it flip the mobile market on its head, eliminating or sidelining even some of the strongest players, but in the embedded space it is likely going to become the de facto standard UI for a vast majority of user-centric embedded devices. There are even signs that it might displace classic "embedded Linux" in headless (non-user-centric) devices.

An entire ecosystem is therefore rapidly building around Android. Silicon and System-on-Chip (SoC) manufacturers such as ARM, TI, Qualcomm, Freescale, and Nvidia have added Android support for their products, and handset and tablet manufacturers such as Motorola, Samsung, HTC, Sony, LG, Archos, Dell, and ASUS ship an ever-increasing number of Android-equipped devices. This ecosystem also includes a growing number of diverse players, such as Amazon, Verizon, Sprint, and Barnes & Noble, creating their own application markets.

Grassroots communities and projects are also starting to sprout around Android, even though it is developed behind closed doors. Many of those efforts are done using public mailing lists and forums, like classic open source projects. Such community efforts typically start by forking the official Android source releases to create their own Android distributions with custom features and enhancements. Such is the case, for instance, with the CyanogenMod project, which provides aftermarket images for power users. There are also efforts by various silicon vendors to provide Android versions enabled or enhanced for their platforms. For example, Linaro—a nonprofit organization created by ARM SoC vendors to consolidate their platform-enablement work—provides its own optimized Android tree. Other efforts follow in the footsteps of phone modders, which essentially rely on hacking the binaries provided by the manufacturers to create their own modifications or variants. Have a look at Appendix E for a full list of AOSP forks and the communities developing them.

A Word on the Open Handset Alliance

As I mentioned earlier, the OHA was the initial vehicle through which Android was first announced. It describes itself on its website as "a group of 82 technology and mobile companies who have come together to accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience. Together we have developed Android, the first complete, open, and free mobile platform."

Beyond the initial announcement, however, it is unclear what role the OHA plays. For example, an attendee at the "Fireside Chat with the Android Team" at Google I/O 2010 asked the panel what privileges were conferred to him as a developer for belonging to a company that is part of the OHA. After asking around the panel, the speaker essentially answered that the panel didn't know because they aren't the OHA. Hence, it would

appear that OHA membership benefits are not clear to the Android development team itself.

The role of the OHA is further blurred by the fact that it does not seem to be a full-time organization with board members and permanent staff. Instead, it's just an "alliance." In addition, there is no mention of the OHA within any of Google's Android announcements, nor do any new Android announcements emanate from the OHA. In sum, one would be tempted to speculate that Google likely put the OHA together mainly as a marketing front to show the industry's support for Android, but that in practice it has little to no bearing on Android's development.

Getting "Android"

There are two main pieces required to get Android working on your embedded system: an Android-compatible Linux kernel and the Android Platform.

For a very long time, getting an Android-compatible Linux kernel was a difficult task; it continues to be in some cases at the time of this writing. Instead of using a "vanilla" kernel from *http://kernel.org* to run the Platform, you needed either to use one of the kernels available within the AOSP or to patch a vanilla kernel to make it Android-compatible. The underlying issue was that many additions were made to the kernel by the Android developers in order to allow their custom Platform to work. In turn, these additions' inclusion in the official mainline kernel were historically met with a lot of resistance.

While we'll discuss kernel issues in greater detail in the next chapter, know that starting from the Kernel Summit of 2011 in Prague, the kernel developers decided to proactively seek to mainline the features required to run the Android Platform on top of the official Linux kernel releases. As such, many of the required features have since been merged, while others have been (or, at the time of this writing, are currently being) replaced or superseded by other mechanisms. At the time of this writing, the easiest way to get yourself an Android-ready kernel was to ask your SoC vendor. Indeed, given Android's popularity, most major SoC vendors provide active support for all Android-required components for their products.

The Android Platform is essentially a custom Linux distribution containing the user-space packages that make up what is typically called "Android." The releases listed in Table 1-1 are actually Platform releases. We will discuss the content and architecture of the Platform in the next chapter. For the time being, keep in mind that a Platform release has a role similar to that of standard Linux distributions such as Ubuntu or Fedora. It's a self-coherent set of software packages that, once built, provides a specific user experience with specific tools, interfaces, and developer APIs.

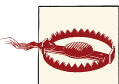
While the proper term to identify the source code corresponding to the Android distribution running on top of an Android-compatible kernel is "Android Platform," it is commonly referred to as "the AOSP"—as is the case in fact throughout this book—even though the Android Open Source Project proper, which is hosted on this site, contains a few more components in addition to the Platform, such as sample Linux kernel trees and additional packages that would not typically be downloaded when the Platform is fetched using the usual *repo* command.

Hacking Binaries

Lack of access to Android sources hasn't discouraged passionate modders from actually hacking and customizing Android to their liking. For example, the fact that Android 3.x/Honeycomb wasn't available didn't preclude modders from getting it to run on the Barnes & Noble Nook. They achieved this by retrieving the executable binaries found in the emulator image provided as part of the Honeycomb SDK and used those as is on the Nook, albeit forfeiting hardware acceleration. The same type of hack has been used to "root" or update versions of various Android components on actual devices for which the manufacturer provides no source code.

Legal Framework

Like any other piece of software, Android's use and distribution is limited by a set of licenses, intellectual property restrictions, and market realities. Let's look at a few of these.



Obviously I'm not a lawyer and this isn't legal advice. You should talk to competent legal counsel to see how any of the applicable terms or licenses apply to your specific case. Still, I've been around open source software long enough that you could consider what follows as an engineer's educated point of view.

Code Licenses

As we discussed earlier, there are two parts to "Android": an Android-compatible Linux kernel and an AOSP release. Even though it's modified to run the AOSP, the Linux kernel continues to be subject to the same GNU GPLv2 license that it has always been under. As such, remember that you are not allowed to distribute any modifications you make to the kernel under any other license than the GPL. Hence, if you take a kernel version from *http://android.googlesource.com* or your SoC vendor and modify it to make it run on your system, you are allowed to distribute the resulting kernel image in your product

only so long as you abide by the GPL. This means you must make the sources used to create the image, including your modifications, available to recipients under the terms of the GPL.

The *COPYING* file in the kernel's sources includes a notice by Linus Torvalds that clearly identifies that only the kernel is subject to the GPL, and that applications running on top of it are **not** considered "derived works." Hence, you are free to create applications that run on top of the Linux kernel and distribute them under the license of your choice.

These rules and their applicability are generally well understood and accepted within open source circles and by most companies that opt to support the Linux kernel or to use it as the basis for their products. In addition to the kernel, a large number of key components of Linux-based distributions are typically licensed under one form or another of the GPL. The GNU C library (glibc) and the GNU compiler (GCC), for example, are licensed under the LGPL and the GPL respectively. Important packages commonly used in embedded Linux systems such as uClibc and BusyBox are also licensed under the LGPL and the GPL.

Not everyone is comfortable with the GNU GPL, however. Indeed, the restrictions it imposes on the licensing of derived works can pose a serious challenge to large organizations, especially given geographic distribution, cultural differences among the various locations of development subunits, and the reliance on external subcontractors. A manufacturer selling a product in North America, for example, might have to deal with dozens, if not hundreds, of suppliers to get that product to the market. Each of these suppliers might deliver a piece that may or may not contain GPL'ed code. Yet the manufacturer whose name appears on the item sold to the customer will be bound to provide the sources to the GPL components regardless of which supplier originated them. In addition, processes must be put in place to ensure that engineers who work on GPL-based projects are abiding by the licenses.

When Google set out to work with manufacturers on its "open" phone OS, therefore, it appears that very rapidly it became clear that the GPL had to be avoided as much as possible. In fact, other kernels than Linux were apparently considered, but Linux was chosen because it already had strong industry support, particularly from ARM silicon manufacturers, and because it was fairly well isolated from the rest of the system, so that its GPL licensing would have little impact.[6]

It was decided, though, that every effort would be made to make sure that the vast majority of user-space components would be based on licenses that did not pose the same logistical issues as the GPL. That is why many of the common GPL- and LGPL-licensed components typically found in embedded Linux systems, such as glibc, uClibc,

6. See this LWN post by Brian Swetland, a member of Android's kernel development team, for more information on the rationale behind these choices.

and BusyBox, aren't included in the AOSP. Instead, the bulk of the components created by Google for the AOSP are published under the Apache License 2.0 (a.k.a. ASL) with some key components, such as the Bionic library (a replacement for glibc and uClibc) and the Toolbox utility (a replacement for BusyBox), licensed under the BSD license. Some classic open source projects are also incorporated, mostly as is in source form under their original licensing, into the AOSP within the *external/* directory. This means that parts of the AOSP are made of software that is neither ASL nor BSD. The AOSP does, in fact, still contain GPL and LGPL components. The distribution of the binaries resulting from the compiling of such components, however, should not pose any problems since they aren't meant to be typically customized by the OEM (i.e., no derived works are expected to be created) and the original sources of those components as used in the AOSP are readily available for all to download at *http://android.googlesource.com*, thereby complying, where necessary, with the GPL's requirement that redistribution of derivative works continue being made under the GPL.

Unlike the GPL, the ASL does not require that derivative works be published under a specific license. In fact, you can choose whatever license best suits your needs for the modifications you make. Here are the relevant portions from the ASL (the full license is available from the Apache Software Foundation):

- "Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form."
- "You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License."

Furthermore, the ASL explicitly provides a patent license grant, meaning that you do not require any patent license from Google for using the ASL-licensed Android code. It also imposes a few "administrative" requirements—such as the need to clearly mark modified files, to provide recipients with a copy of the ASL license, and to preserve *NOTICE* files as is. Essentially, though, you are free to license your modifications under the terms that fit your purpose. The BSD license that covers Bionic and Toolbox allows similar binary-only distribution.

Hence, manufacturers can take the AOSP and customize it to their needs while keeping those modifications proprietary if they wish, so long as they continue abiding by the rest of the provisions of the ASL. If nothing else, this diminishes the burden of having

to implement a process to track all modifications in order to provide those modifications back to recipients who would be entitled to request them had the GPL been used instead.

Adding GPL-Licensed Components

Although every effort has been made to keep the GPL out of Android's user-space as much as possible, there are cases where you may want to explicitly add GPL-licensed components to your Android distribution. For example, you want to include either glibc or uClibc, which are POSIX-compliant C libraries—in contrast to Android's Bionic, which is not—because you would like to run preexisting Linux applications on Android without having to port them over to Bionic. Or you may want to use BusyBox in addition to Toolbox, since the latter is much more limited in functionality than the former.

These additions may be specific to your development environment and may be removed in the final product, or they may be permanent fixtures of your own customized Android. No matter which avenue you decide on, whether it be plain Android or Android with some additional GPL packages, remember that you must follow the licenses' requirements.

Branding Use

While being very generous with Android's source code, Google controls most Android-related branding elements more strictly. Let's take a look at some of those elements and their associated terms of use. For the official list, along with the official terms, have a look at this site.

Android robot
: This is the familiar green robot seen everywhere around all things Android. Its role is similar to the Linux penguin, and the permissions for its use are similarly permissive. In fact, Google states that it "can be used, reproduced, and modified freely in marketing communications." The only requirement is that proper attribution be made according to the terms of the Creative Commons Attribution license.

Android logo
: This is the set of letters in custom typeface that spell out A-N-D-R-O-I-D and that appear during the device and emulator bootup, and on the Android website. You are not authorized to use that logo under any circumstance. Chapter 7 shows you how to replace the bootup logo.

Android custom typeface
: This is the custom typeface used to render the Android logo, and its use is as restricted as the logo.

"Android" in official names and messaging

As Google states, " 'Android' by itself cannot be used in the name of an application name or accessory product. Instead use 'for Android.' " Therefore, you can't say "Android MediaPlayer," but you can say "MediaPlayer for Android." Google also states that "Android may be used as a descriptor, as long as it is followed by a proper generic term" such as "Android™ application" for example. Of course, proper trademark attribution must always be made. In sum, you can't name your product "Android Foo" without Google's permission, though "Foo for Android" is fine.

"Android"-branded devices

As the FAQ for the Android Compatibility Program (ACP) states: "[I]f a manufacturer wishes to use the Android name with their product...they must first demonstrate that the device is compatible." Branding your device as being "Android" is therefore a privilege that Google intends to police. In essence, you will have to make sure your device is compliant and then talk to Google and enter into some kind of agreement with it before you can advertise your device as being "Foo Android." We will cover the Android Compatibility Program later in this chapter.

"Droid" in official names

You may not use "Droid" alone in a name, such as "Foo Droid," for example. For some reason the I haven't yet entirely figured out, "Droid" is a trademark of Lucasfilm. Achieve a Jedi rank, you likely must, before you can use it.

Word (and Brand) Play

While Google holds strict control over the use of the Android brand, the ASL used for licensing the bulk of the AOSP states the following: "This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the *NOTICE* file."

While this clearly says you have no right to use the associated trademark, the "reasonable and customary use in describing the origin" exception is seen by many as allowing you to state that your device is "AOSP based." Some push this further and simply state that their product is "based on Android" or "Android based." You'll even find some clever marketing material sporting the Android robot to advertise a product without mentioning the word "Android."

Probably one of the sneakiest wordplays I've seen is when a product lists the following as part of one of its features: "Runs Android applications." You can bet yourself a couple of green robots that if it runs Android applications, it's almost guaranteed to contain the AOSP in some way, shape, or form.

Google's Own Android Apps

While the AOSP contains a core set of applications that are available under the ASL, "Android"-branded phones usually contain an additional set of "Google" applications that are not part of the AOSP, such as Play Store (the "app market" app), YouTube, "Maps and Navigation," Gmail, etc. Obviously, users expect to have these apps as part of Android, and you might therefore want to make them available on your device. If that is the case, you will need to abide by the ACP and enter into an agreement with Google, very much in line with what you need to do to be allowed to use "Android" in your product's name. We will cover the ACP shortly.

Alternative App Markets

Though the main app market (i.e., Google Play) is the one hosted by Google and made available to users through the Play Store app installed on "Android"-branded devices, other players are leveraging Android's open APIs and open source licensing to offer alternative app markets. Such is the case with online merchants such as Amazon and Barnes & Noble, as well as mobile network operators such as Verizon and Sprint. In fact, I know of nothing that would preclude you from creating your own app store. There is even at least one open source project, the Affero-licensed F-Droid Repository, that provides both an app market application and a corresponding server backend under the GPL.

Oracle versus Google

As part of acquiring Sun Microsystems, Oracle also acquired Sun's intellectual property (IP) rights to the Java language and, according to Java creator James Gosling,[7] it was clear during the acquisition process that Oracle intended from the outset to go after Google with Sun's Java IP portfolio. And in August 2010 it did just that, filing suit against Google, claiming that it infringed on several patents and committed copyright violations.

Without going into the merits of the case, it's obvious that Android does indeed heavily rely on Java. And clearly Sun created Java and owned a lot of intellectual property around the language it created. In what appears to have been an effort to anticipate any claims Sun may put forward against Android, the Android development team went out of its way to use as little of Sun's Java in the Android OS as possible. Java is in fact composed mainly of three things: the language and its semantics, the virtual machine that runs the Java byte-code generated by the Java compiler, and the class library that contains the packages used by Java applications at runtime.

7. See Gosling's blog postings on the topic at *http://nighthacks.com/roller/jag/entry/the_shit_finally_hits_the* and *http://nighthacks.com/roller/jag/entry/quite_the_firestorm* for more details.

The official versions of the Java components are provided by Oracle as part of the Java Development Kit (JDK) and the Java Runtime Environment (JRE). Android, on the other hand, relies only on the Java compiler found in the JDK for building parts of the AOSP; that compiler isn't included as part of the images generated by the AOSP. Also, instead of using Oracle's Java VM, Android relies on Dalvik, a VM custom built for Android, and instead of using the official class library, Android relies on Apache Harmony, a clean-room reimplementation of the class library. Hence, it would seem that Google made every reasonable effort to at least avoid any copyright and/or distribution issues.

Still, it remains to be seen where these legal proceedings will go. Although by May 2012 Google had prevailed on both the copyright and patent fronts of the initial trial, Oracle appealed the verdict in October of that same year. There is of course a lot at stake, and it will likely take many years for this saga to play itself out. If you want to follow the latest round of these proceedings or read up on past episodes, I suggest you have a look at the Groklaw website and consult the relevant Wikipedia entry.

Another indirectly related, yet very relevant, development is that IBM joined Oracle's OpenJDK efforts in October 2010. IBM had been the driving force behind the Apache Harmony project, which is the class library used in Android, and its departure pretty much ensures that the project will become orphaned. How this development impacts Android is unknown at the time of this writing.

Incidentally, though he later left, James Gosling joined Google in March 2011.

Mobile Patent Warfare

The previous section is to some extent but the tip of the iceberg with regard to litigation and legal wranglings ongoing in the mobile world at the time of this writing. Sales of mobile phones have overtaken the sales of traditional PCs, and the mobile market's growth has resulted in the majority of players being somehow involved in legal maneuvers against and/or because of its competitors. There's even a Wikipedia entry entitled Smartphone wars dedicated to listing the ongoing battles.

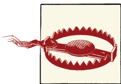
It's hard to say where any of this will go. There seems to be no end to the variety of strategies companies will employ or the lengths to which they'll go to ensure they prevail. Apple and Samsung, for instance, are at the time of this writing involved in court cases against each other in quite a few countries. Microsoft is also rumored to be contacting various manufacturers to request royalties for the use of Android; as evidenced by some of the filings made by Barnes & Noble with the courts after it was sued by Microsoft for refusing to pay.

How any of this might affect your own product is difficult to say. As always, consult with competent legal counsel as needed. Usually it's a question of volume. So if your product is for a niche market, you're probably too small a fish to matter. If you're creating a mass-market product, on the other hand, you'll likely want to make sure you've covered all your bases.

Hardware and Compliance Requirements

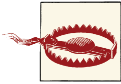
In principle, Android should run on any hardware that runs Linux. Android has in fact been made to run on ARM, x86, MIPS, SuperH, and PowerPC—all architectures supported by Linux. A corollary to this is that if you want to port Android to your hardware, you must first port Linux to it. Beyond being able to run Linux, though, there are few other hardware requirements for running the AOSP, apart from the logical requirement of having some kind of display and pointer mechanism to allow users to interact with the interface. Obviously, you might have to modify the AOSP to make it work on your hardware configuration, if you don't support a peripheral it expects. For instance, if you don't have a GPS unit in your product, you might want to provide a mock GPS HAL module, as the Android emulator does, to the AOSP. You will also need to make sure you have enough memory to store the Android images and a sufficiently powerful CPU to give the user a decent experience.

In sum, therefore, there are few restrictions if you just want to get the AOSP up and running on your hardware. If, however, you are working on a device that must carry "Android" branding or must include the standard Google-owned applications found in typical consumer Android devices—such as the Maps or Play Store applications—you need to go through the Android Compatibility Program (ACP) mentioned earlier. There are two separate yet complementary parts to the ACP: the Compliance Definition Document (CDD) and the Compliance Test Suite (CTS). Even if you don't intend to participate in the ACP, you might still want to take a look at the CDD and the CTS, as they give a very good idea about the general mind-set that went into the design goals of the Android version you intend to use.



Every Android release has its own CDD and CTS. You must therefore use the CDD and CTS that match the version you intend to use for your final product. If you switch Android releases midway through your project—because, for instance, a new Android release comes out with cool new features you'd like to have—you will need to make sure you comply with that release's CDD and CTS. Keep in mind also that you need to interact with Google to confirm compliance. Hence, switching may involve jumping through a few hoops and potential product delivery delays.

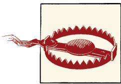
The overarching goal of the ACP, and therefore the CDD and the CTS, is to ensure a uniform ecosystem for users and application developers. Hence, before you are allowed to ship an "Android"-branded device, Google wants to make sure you aren't fragmenting the Android ecosystem by introducing incompatible or crippled products. This, in turn, makes sense for manufacturers since they are benefiting from the compliance of others when they use the "Android" branding. Look at this site for more details about the ACP.



Note that Google reserves the right to decline your participation in the Android ecosystem, and therefore prevent your ability to ship the Play Store app with your device and use the "Android" branding. As stated on their site: "Unfortunately, for a variety of legal and business reasons, we aren't able to automatically license Google Play to all compatible devices."

Compliance Definition Document

The CDD is the policy part of the ACP and is available at the ACP URL above. It specifies the requirements that must be met for a device to be considered compatible. The language in the CDD is based on RFC2119, with a heavy use of "MUST," "SHOULD," "MAY," etc. to describe the different attributes. Around 25 pages in length, it covers all aspects of the device's hardware and software capabilities. Essentially, it goes over every aspect that cannot simply be automatically tested using the CTS. Let's go over some of what the CDD requires.



This discussion is based on the Android 2.3/Gingerbread CDD. The specific version you use will likely have slightly different requirements.

Software

This section lists the Java and native APIs along with the web, virtual machine, and user interface compatibility requirements. Essentially, if you are using the AOSP, you should readily conform to this section of the CDD.

Application packaging compatibility

This section specifies that your device must be able to install and run *.apk* files. All Android apps developed using the Android SDK are compiled into *.apk* files, and these are the files that are distributed through Google Play and installed on users' devices.

Multimedia compatibility

Here the CDD describes the media codecs (decoders and encoders), audio recording, and audio latency requirements for the device. The AOSP includes the StageFright

multimedia framework, and you can therefore conform to the CDD by using the AOSP. However, you should read the audio recording and latency sections, as they contain specific technical information that may impact the type of hardware or hardware configuration your device must be equipped with.

Developer tool compatibility

This section lists the Android-specific tools that must be supported on your device. Basically, these are the common tools used during app development and testing: *adb*, *ddms*, and *monkey*. Typically, developers don't interact with these tools directly. Instead, they usually develop within the Eclipse development environment and use the Android Development Tool (ADT) plug-in, which takes care of interacting with the lower-level tools.

Hardware compatibility

This is probably the most important section for embedded developers, as it likely has profound ramifications on the design decisions made for the targeted device. Here's a summary of what each subsection spells out.

Display and graphics

- Your device's screen must be at least 2.5 inches in physical diagonal size.
- Its density must be at least 100dpi.
- Its aspect ratio must be between 4:3 and 16:9.
- It must support dynamic screen orientation from portrait to landscape and vice versa. If orientation can't be changed, then it must support letterboxing, since apps may force orientation changes.
- It must support OpenGL ES 1.0, though it may omit 2.0 support.

Input devices

- Your device must support the Input Method Framework, which allows developers to create custom onscreen, soft keyboards.
- It must provide at least one soft keyboard.
- It can't include a hardware keyboard that doesn't conform to the API.
- It must provide Home, Menu, and Back buttons.
- It must have a touch screen, whether it be capacitive or resistive.
- It should support independent tracked points (multitouch) if possible.

Sensors

While all sensors are qualified using "SHOULD," meaning that they aren't compulsory, your device must accurately report the presence or absence of sensors and must return an accurate list of supported sensors.

Data connectivity

The most important item here is an explicit statement that Android may be used on devices that don't have telephony hardware. This was added to allow for Android-based tablet devices. Furthermore, your device should have hardware support for 802.11x, Bluetooth, and near field communication (NFC). Ultimately, your device must support some form of networking that permits a bandwidth of 200Kbps.

Cameras

Your device should include a rear-facing camera and may include a front-facing one as well.

Memory and storage

- Your device must have at least 128MB for storing the kernel and user-space.
- It must have at least 150MB for storing user data.
- It must have at least 1GB of "shared storage." This is typically, though not always, the removable SD card.
- It must also provide a mechanism to access shared data from a PC. In other words, when the device is connected through USB, the content of the SD card must be accessible on the PC.

USB

This requirement is likely the one that most heavily demonstrates how user-centric "Android"-branded devices must be, since it essentially assumes that the user owns the device and therefore requires you to allow users to fully control the device when it's connected to a computer. In some cases this might be a showstopper for you, as you may not actually want or may not be able to have users connect your embedded device to a computer. Nevertheless, the CDD requires the following:

- Your device must implement a USB client, connectable through USB-A.
- It must implement the Android Debug Bridge (ADB) protocol as provided in the *adb* command over USB.
- It must implement USB mass storage, thereby allowing the device's SD card to be accessed on the host.

Newer CDDs obviously have evolved from this list. There's no longer a need to have physical Home, Menu, and Back buttons since 3.0, since those can be displayed onscreen. OpenGL ES 2.0 support is also now mandatory. In addition to USB mass storage support, the device can also now provide Media Transfer Protocol (MTP) instead.

Performance compatibility

Although the CDD doesn't specify CPU speed requirements, it does specify app-related time limitations that will impact your choice of CPU speed. For instance:

- The Browser app must launch in less than 1300ms.
- The MMS/SMS app must launch in less than 700ms.
- The AlarmClock app must launch in less than 650ms.
- Relaunching an already-running app must take less time than the original launch.

Security model compatibility

Your device must conform to the security environment enforced by the Android application framework, Dalvik, and the Linux kernel. Specifically, apps must have access and be submitted to the permission model described as part of the SDK's documentation. Apps must also be constrained by the same sandboxing limitations they have by running as separate processes with distinct user IDs (UIDs) in Linux. The filesystem access rights must also conform to those described in the developer documentation. Finally, if you aren't using Dalvik, whatever VM you use should impose the same security behavior as Dalvik.

Software compatibility testing

Your device must pass the CTS, including the human-operated CTS Verifier part. In addition, your device must be able to run specific reference applications from Google Play.

Updatable software

There has to be a mechanism for your device to be updated. This may be done over the air (OTA) with an offline update via reboot. It also may be done using a "tethered" update via a USB connection to a PC, or be done "offline" using removable storage.

Compliance Test Suite

The CTS comes as part of the AOSP, and we will discuss how to build and use it in Chapter 4. The AOSP includes a special build target that generates the *cts* command-line tool, the main interface for controlling the test suite. The CTS relies on *adb* to push and run tests on the USB-connected target. The tests are based on the JUnit Java unit testing framework, and they exercise different parts of the framework, such as the APIs, Dalvik, Intents, Permissions, etc. Once the tests are done, they will generate a ZIP file containing XML files and screenshots that you need to submit to *cts@android.com*.

Development Setup and Tools

There are two separate sets of tools for Android development: those used for application development and those used for platform development. If you want to set up an application development environment, have a look at *Learning Android* or at Google's online documentation. If you want to do platform development, as we will do here, your tool needs will vary, as you will see later in this book.

At the most basic level, though, you need to have a Linux-based workstation to build the AOSP. In fact, at the time of this writing, Google's only supported build environment is 64-bit Ubuntu 10.04. That doesn't mean that another Ubuntu version or even another Linux distribution won't work or, in the case of Android versions up to Gingerbread, that you won't be able to build the AOSP on a 32-bit system,[8] but essentially that configuration reflects Google's own Android compile farms configuration. An easy way to get your hands dirty with AOSP work without changing your workstation OS is to create an Ubuntu virtual machine using your favorite virtualization tool. I typically use VirtualBox, since I've found that it makes it easy to access the host's serial ports in the guest OS.

In some cases, even though 32-bit build support wasn't available for a given Android version, patches were created to make such compiling possible. This is especially true for Gingerbread. So even though the official tree may not support 32-bit builds, you may be able to find another tree that does or a mailing list posting that explains how to do it. Still, it remains that newer AOSP versions require more and more powerful machines to build in a reasonable amount of time, and most of these systems end up being 64 bit. Hence, the impetus for supporting builds on 32-bit systems diminishes with every new version of Android.

No matter what your setup is, keep in mind that the AOSP is several gigabytes in size before building, and its final size is much larger. Gingerbread, for example, is about 3GB in size uncompiled and grows to about 10GB once compiled, while 4.2/Jelly Bean is 6GB uncompiled and grows to about 24GB once compiled.[9] When you factor in that you are likely going to operate on a few separate versions—for testing purposes if for no other reason—you rapidly realize that you'll need tens if not hundreds of gigabytes for serious AOSP work. Also note that during the period this book was written (2011 to 2013), build times for the latest AOSP on the highest-end machines have always hovered

8. More recent versions such as JellyBean 4.1 and 4.2 can be built only on 64-bit systems.

9. These uncompiled numbers don't count the space taken by the *.git* and *.repo* directories in the tree. The uncompiled size of 2.3.7/Gingerbread with those directories is 5.5GB and that of 4.2/Jelly Bean is 18GB.

between 30 minutes to an hour. Even minor modifications may result in a five-minute run to complete the build or regenerate output images. You will therefore also likely want to make sure you have a fairly powerful machine when developing Android-based embedded systems. We'll discuss the AOSP build and its requirements in greater detail in Chapter 4.