An Introduction to Android Firmware

Ashraff Hathibelagal Jul 4, 2016

9 mins | English

Android phones and tablets are generally a lot more open than their counterparts running operating systems such as **iOS**, **Tizen**, or **Windows 10 Mobile**. If you don't like the firmware the device manufacturer has installed on your Android device, you are free to replace it with your own custom firmware. **CyanogenMod**, **Paranoid Android**, and the **Pure Nexus Project** are examples of custom firmware that enjoy a lot of popularity among Android users.

Custom firmware is also the only way you can install newer versions of Android on devices that are no longer supported by their manufacturers. Unless you own a device that belongs to the **Nexus** or **Android One** series, I'm sure you knew that already.

In this article, I help you understand what Android firmware really is and how an Android device uses it. I also introduce you to the tools you can use to replace a device's firmware.

A Word of Caution

Replacing firmware is a risky operation that can potentially make your device unusable. In

most cases it also voids your device's warranty. Make sure that you have a backup of your data and a copy of your device's factory image handy before you go ahead and experiment with flashing custom firmware.

1. What Is Android Firmware?

Originally, firmware was a term used to refer to tiny, mission-critical programs installed in the read-only memory, or ROM, of an electronic device. Modifying firmware was either impossible or required special equipment that was usually out of the reach of ordinary end users.

Android firmware, however, is very different. It includes the entire Android operating system and it is stored in a writable form of memory called NAND flash memory, the same type of memory that is used in storage devices, such as USB sticks and SD cards. The word **firmware** is used only because device manufacturers didn't bother to come up with a new word for it.

Android firmware is also often referred to as **Android ROM** because, by default, it is not possible for users to directly write to it.

2. What Does Android Firmware Contain?

Firmware installed on an Android device by its manufacturer contains a build of the Android operating system and two additional closed source programs that are usually irreplaceable, a **bootloader** and **radio firmware**.

Understanding Bootloaders

An Android bootloader is a small piece of proprietary code that is responsible for starting the Android operating system when an Android device is powered on. However, the bootloader almost always performs one more task. It checks if the operating system it is starting is authentic.

How does it decide what is authentic? It checks if the boot partition has been signed using a unique **OEM key**, which is short for Original Equipment Manufacturer key. The OEM key, of course, belongs to the device manufacturer, is private, and there is no way you can know what it is.

Because of the authenticity check, you cannot directly install a custom ROM on an Android device. Thankfully, these days, most device manufacturers allow users to disable the check. In Android jargon, they allow users to **unlock** the bootloader.

The exact procedure you need to follow in order to unlock the bootloader depends on your device. Some manufacturers, such as Sony and HTC, expect you to provide a secret unlock token. Others just expect you to run a fixed set of commands using a terminal.

Usually, a tool called **fastboot**, which is a part of the Android SDK, is used to run the unlock commands. For example, if you own a Nexus device, you can unlock its bootloader by running the following command:

```
1 fastboot flashing unlock
```

```
fastboot flashing unlock
```

You learn more about **fastboot** later in this article. Note that, if you own a device that has a bootloader that cannot be unlocked, there is no easy way for you to modify or replace its firmware.

Understanding Radio Firmware

It might come as a surprise to you, but your Android smartphone actually runs another operating system on an independent processor called a **baseband processor**. **Radio firmware** refers to the operating system that runs on the baseband processor.

Usually, it is an RTOS, which short for **real-time operating system**, and is responsible for managing the cellular radio capabilities of the device. In other words, it is what allows your device to make calls and connect to the internet using wireless technologies such 2G, 3G, and 4G LTE.

The RTOS is a proprietary piece of code and popular baseband processor manufacturers, such as Qualcomm, MediaTek, and Spreadtrum, make sure that its internal workings stay a secret. The Android operating system usually communicates with the RTOS using sockets and callbacks.

Generally, it is not a good idea to replace the radio firmware of your device.

Understanding Android Builds

The Android build is the only part of the firmware that is created from open source code. Consequently, this is the only part that you can modify and extend. When you hear Android enthusiasts say "I flashed a new ROM on my device", you can be sure that they are talking about a new Android build.

An Android build is usually shared in the form of a ZIP file that can be used by **fastboot**. It has the following contents:

```
update.zip
```

```
update.zip
|-- android-info.txt
|-- boot.img
|-- recovery.img
|-- system.img
`-- userdata.img
```

android-info.txt is a text file specifying the prerequisites of the build. For example, it could specify the version numbers of the bootloader and the radio firmware that the build needs. Here is a sample **android-info.txt** file:

```
require board=herring
require version-bootloader=I9020XXJK1
require version-baseband=I9020XXKD1
```

boot.img is a binary file that contains both a Linux kernel and a ramdisk in the form of a GZIP archive. The kernel is a boot executable zImage that can be used by the bootloader.

The ramdisk, on the other hand, is a read-only filesystem that is mounted by the kernel during the boot process. It contains the well known **init** process, the first process started by any Linux-based operating system. It also contains various daemons such as **adbd** and **healthd**, which are started by the **init** process. Here is what the directory tree of the **ramdisk** looks like:

```
ramdisk/
|-- charger -> /sbin/healthd
|-- data
|-- default.prop
|-- dev
|-- file_contexts
|-- fstab.grouper
|-- init
|-- init.environ.rc
|-- init.grouper.rc
|-- init.grouper.usb.rc
|-- init.rc
|-- init.recovery.grouper.rc
|-- init.trace.rc
|-- init.usb.rc
|-- init.zygote32.rc
|-- proc
|-- property_contexts
|-- sbin
|   |-- adbd
|   |-- healthd
```

```
|   |-- healthd
|   |-- ueventd -> ../init
|   `-- watchdogd -> ../init
|-- seapp_contexts
|-- selinux_version
|-- sepolicy
|-- service_contexts

|-- sys
|-- system
|-- ueventd.grouper.rc
`-- ueventd.rc
```

system.img is the partition image that will be mounted on the empty **system** directory you can see in the above tree. It contains the binaries required for the Android operating system to run. It includes the system apps, fonts, framework JAR files, libraries, media codecs, and more. Obviously, this is the file Android users are most interested in when they flash a new ROM.

The system image is also the file that makes most Android users develop an interest in flashing custom firmware. System image files provided by device manufacturers are often full of unnecessary apps and customizations, informally called bloatware. The only way to remove the bloatware is to replace the manufacturer's system image with a more desirable system image.

userdata.img is a partition image that will be mounted on the empty **data** directory you can see in the ramdisk directory tree. When you download a custom ROM, this image is usually blank and it is used to reset the contents of the **data** directory.

recovery.img is very similar to **boot.img**. It has a boot executable kernel file the bootloader can use and a ramdisk. Consequently, the recovery image too can be used to start an Android device. When it is used, instead of Android, a very limited operating system is started that allows the user to perform administrative operations, such as resetting the device's user data, installing new firmware, and creating backups.

The procedure you need to follow in order to boot up using the recovery image is device specific. Usually, it involves entering the bootloader mode, also called **fastboot mode**, by pressing a combination of hardware keys present on the device, and then selecting the **Recovery** option. For example, on a Nexus device you need to press and hold the power

button in combination with the volume down button.

Alternatively, you can use **adb**, a tool included in the Android SDK, to directly enter recovery mode.

```
adb reboot recovery
```

3. Using fastboot

The easiest way to flash new firmware on your device is to use the **fastboot** tool. **fastboot** follows the fastboot protocol to communicate with an Android device. However, it can only do this when the device has been started in fastboot mode. The quickest way to enter fastboot mode is by using **adb**:

```
adb reboot bootloader
```

To flash a custom ROM that is available in the form of a ZIP file containing all the image files I mentioned in the previous section, you can use the `fastboot update` command. For example, here is how you would flash a ROM present in a file called **update.zip**:

```
fastboot update update.zip
```

If you want to flash only a specific image, you can do so using the `fastboot flash` command. For example, here is how you would flash only the system image:

```
fastboot flash system system.img
```

Similarly, if you want to replace only the boot image, you would use the following command:

```
fastboot flash boot boot.img
```

```
fastboot flash boot boot.img
```

It is always a good idea to test if a boot or recovery image is working before actually flashing it to your device. To do so, you can use the `fastboot boot` command. For example,

here is how you would check if a custom recovery image called **twrp.img** is compatible with your device:

```
fastboot boot twrp.img
```

Note that none of the **fastboot** commands I mentioned in this section will work if the bootloader of your device has not been unlocked.

Conclusion

You now know what Android firmware is and how to replace it. I want you to understand that replacing firmware is a risky operation that can potentially make your device unusable. In most cases it also voids your device's warranty. Make sure that you have a backup of your data and a copy of your device's factory image handy before you go ahead and experiment with flashing custom firmware.

Android SDK CLIs Mobile Development How-To

Did you find this post useful?

Yes No

Want a weekly email summary?

Subscribe below and we'll send you a weekly email summary of all new Code tutorials. Never miss out on learning about the next big thing.

Sign up

Ashraff Hathibelagal

Hathibelagal is an independent Android app developer and **blogger** who loves tinkering with new frameworks, SDKs, and devices.

hathibel

HELP

tuts+

30,328 Tutorials

1,316 Courses

50,287 Translations

Certified B Corporation