Unboxing Android USB

A hands on approach with real world examples

Rajaram Regupathy

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

The Android open platform, which was introduced in 2007, is now in more than 50 million devices. The application store statistics show billions of downloads. It has literally conquered the mobile handset market, overtaking many established players. It is also expanding beyond mobile platforms into unique products such as the Android Stick, which converts a normal TV to a smart one.

If you are a developer who works on embedded systems, there is no escape from this ever-growing platform. This inevitability creates a need for good reference books for engineers who are interested in getting started with Android. There are many books in the market covering Android application programming and its development environment. If you are looking for something like that in this book, you are in the wrong place. This book is much more than that. The book explains the complete Android framework, from the API to the internals of Android, along with the kernel below them.

This book exclusively covers the internals of the Android USB framework. Why USB? Similar to the Android platform, USB is also inevitable in the embedded world. On the Android platform, USB is the primary connectivity solution, as an interface used to debug and also as an interface used to charge the batteries of the Android device.

Does this mean this book is only for USB engineers? In fact, it will be useful to any developer working on the Android platform. Why?

If you are a multimedia developer on the Android platform, you need USB for media transfer or to play back audio. This book explores MTP and USB audio in both USB device and USB host modes.

If you are a core developer who works on charging, you need to understand the USB charging specifications, which are explained in the book.

If you are a networking developer interested in tethering, USB plays a role using the RNDIS specification, which is explained in the book.

If you are an application developer interested in managing USB devices from an Android platform, this book explores the Android USB Service framework, which manages USB functionalities.

Last but not least, Android Debug Bridge (ADB), the debugging tool of Android, is over USB and knowledge of its internals is a definite value-add for any application or platform developer. This book details the internals of ABD to the kernel level.

This book covers everything about USB on Android, from the different USB classes supported in device mode to the USB host framework that manages the USB devices connected to the Android platform. Each chapter explains USB class specification before exploring how the functionality (class) is implemented on the Android platform. This gives readers a clean perspective as to what the USB specification demands and how it is implemented in Android.

The Android framework has migrated to different versions by now. As a platform or application developer, it's important you know about the major changes each version introduced. The book covers the major changes in the USB framework between the versions, including interesting bug fixes that were undocumented in the Android specifications.

Intended Audience

The primary audience for this book are application developers and engineers who work hands-on with Android. This book is for an application developer who has an idea for a USB app and wonders how to implement it. This book will be a definite guide for the developer to manage USB on Android.

Because the book covers APIs to the Linux kernel, core platform developers will find it easy to put data point to debug. Thus, core Android platform developers working on USB, audio, media, and others are the next primary audience for the book.

Technical managers, architects, and senior managers who look for the eagle-eye view of a system are a secondary audience for the book. The book will enable them to understand the different blocks of the Android USB subsystem and help estimate the complexity involved.

Student and engineers can use this book as a do-it-yourself reference, as it explains the different blocks of the Android USB framework, from the application level to the kernel.

What You'll Learn

Understand the Android USB framework, from the APIs to the kernel layer, and enable advanced USB application development.

Learn all the major USB functionalities by exploring the USB class specifications not covered in any of the USB books.

Learn the newly introduced Android Open Accessory (AOA) protocol and explore the developing NFC reader using the AOA protocol.

Learn about critical changes in the Android USB framework among different Android versions.

Learn how USB charging works, with an explanation of the USB battery specification.

Learn how to switch between MTP and mass storage and vice versa, in order to share storage with a host PC.

Salient Features

Real-world useful applications enhance your Android experience, including reverse tethering, AOA audio, AOA NFC reader, switching between MTP and UMS, and more. Complete project source is available, which will help you try it on your own.

Covers advanced technical topics (Android and USB) that aren't covered in other texts.

All design diagrams (Microsoft Visio) are on the CD for reuse by developers and architects.

Covers the major differences in the Android USB framework between Android versions.

Covers all major USB functions, such as MTP, audio, charging, and mass storage, along with Google-defined USB functions like ADB and AOA, all by exploring their specifications.

Chapter Introduction

Though there are different types of Android-powered devices, this book details the Android USB framework with a mobile hand-held device in mind. The following section provides a brief description of each chapter in this book.

Getting Started: The Android USB Framework

Android defines its requirement through the Compatibility Definition Document (CDD) and mandates that Android devices comply with this specification. This chapter provides a brief overview of the USB requirements defined in the Android CDD. The chapter subsequently explains various USB-related Android APIs that the Android framework exports for application developers in order to manage USB functionalities or devices.

Discovering and Managing USB Within Android

Discovering and managing a device is the first step and a crucial part any programming activity. This chapter describes how USB function discovery is made inside the Android framework when an Android device is connected in USB device mode. The chapter also details how a USB device is detected inside the Android framework when an Android device is connected in host mode.

USB Storage

Media is one of the key features of mobile devices and is predominantly managed using USB. Media over USB is managed using two USB specifications: Media Transfer Protocol (MTP) and Mass Storage Class (UMS). This chapter briefly details these two specifications and provides an overview of the USB specification's requirements. The chapter also details how media files are transferred to a host PC when the Android device is in USB device mode (both UMS and MTP).

This chapter also explains how a USB-based external media device (say, a USB flash drive or an MTP device) is managed by the Android framework in USB host mode.

USB Tethering

Tethering is a method by which mobile devices shares their Internet connectivity with other devices, such as personal computers or laptops. An Android device uses the RNDIS protocol over USB to tether and share Internet connectivity with other devices. The RNDIS protocol is Microsoft-specific and is very similar to the USB ECM class specification. This chapter provides a brief overview of the RNDIS specification and explains the USB part of the Android framework that facilitates tethering.

USB Accessory

Android Open Accessory (AOA), an Android-specific class defined by Google, was introduced in the Ice Cream Sandwich version of Android to facilitate Android devices in managing external devices. The chapter details the AOA protocol and its operations with an example application. With the Jelly Bean version of Android, the AOA protocol was improved to support the USB Human Interface Device (HID) class. The chapter provides a brief overview of the USB HID class and its implementation inside the Android framework.

USB Audio

The USB audio specification defines transport that provides an efficient way to propagate and control digital audio. With the Jelly Bean version of Android, an Android system in USB device mode supports the USB audio class. This support of digital audio over USB is packed with the AOA protocol. This chapter provides a brief overview of USB audio specification and subsequently explains the Android framework that implements the device audio class. The chapter explains the device and host audio implementations within the Android framework.

Android Debug Bridge

Android Debug Bridge (ADB) is a command-line client/server debug tool that allows you to communicate with an Android-powered device using USB as a transport. This chapter details the ADB protocol defined by Google and subsequently explains how the Android USB framework implements the ADB protocol.

Appendix A: Battery Charging Using USB

Most battery-powered hand-held devices use a USB port to generate power for charging the battery. Android-powered hand-held devices also use USB as the primary power source to charge the battery. This USB class is covered as part of this appendix since there is no real Android USB framework for battery management. This is because USB charging specification focuses on the charging current and other low-level details; there is no USB-level protocol. This chapter provides a brief overview of the USB charging specification and subsequently explains the USB part of the Android battery manager framework.

Appendix B: Using libusb in Android

Protocols like USB allow developers to write driver at user space to manage its functionality. The USB user space driver called libusb is available in almost all popular desktop operating systems. Since libusb is a generic driver, it can be used with any USB device. This chapter explores how to write a simple application over libusb on the Android platform.

Chapter 1

Getting Started: The Android USB Framework

What you will learn:

- Android USB CDD requirements
- Overview of Android USB packages
- Architectural diagram of Android USB framework
- Android USB APIs

Android has become one of the most successful open platforms, powering up millions of mobile devices and similar embedded devices worldwide. According to Google, more than a million new Android devices are added to this statistic every day. This large market presence and continuous market penetration makes it the ideal platform for developers, SMEs, and bigger enterprises to portray their presence and reach out to end users. For Android devices, Google provides the necessary infrastructure to develop new applications. These devices can reach millions of end users through Google's open market platform named "Google Play."

Such a large development and deployment process necessitates standardization in order to ensure compatibility of these applications across the multitudes of Android devices that exist. To facilitate this, Google created a compatibility program that enables application developers, end users, and platform manufacturers to maintain program consistency and a similar user experience across devices. A detailed overview of the compatibility program is available on Google's Android web site at https://source.android.com/compatibility/overview.html. The compatibility program consists of three key components: Compatibility

Definition Document (CDD), Android Platform Source Code, and a Compatibility Test Suite (CTS). Any device that claims to be an "Android" device has to comply with the Android CDD and successfully pass all CTS test suites.

In order to study the framework within Android, it is important to understand the aforementioned three key components. Thus, in order to best study the Android USB framework, it is important to focus and explore what Android CDD defines as a USB requirement, and how that requirement is implemented.

This chapter starts with exploring the USB section of the Android CDD, and subsequently presents a complete overview of the Android USB framework by providing a break down of the implementation process. Later on, the chapter will explore various USB APIs that the Android framework exports in order to assist an application developer in managing the USB functionality of an Android device.

Android CDD – USB

At the time of this writing, Android 4.4 Kit Kat is the latest version of Android and Android 4.4 CDD defines the compatibility requirement of the Android Kit Kat version. You can find the complete list of Android CDDs on Google's Android website at http://source.android.com/compatibility/downloads.html. So, what is an Android CDD? In simple terms, the Android CDD defines the requirements that must be met in order for a device to claim that it is an Android-compatible device. To an extent, Android CDD is brief in that it is a 30-40 page document. This document can point to specifications like the USB Audio, for example, to indicate the user's expectation. The CDD also identifies features as "must," "must not," "required," "shall," "shall not," "should," "should not," "recommended," "may," and "optional," as per the IETF standard that is defined in RFC2119. It is important for developers to pay attention to these terms and take care while developing Android applications when using an optional feature or any feature listed as "may."

When it comes to USB, an Android device can operate in two modes—USB device mode or USB host mode.

USB Device Mode

When an Android device is connected to a host PC using USB, as illustrated in Figure 1-1, the Android device is said to be in USB device mode and power is sourced from the host PC USB port. (A device that needs more power than the host can provide should have its own power source.)

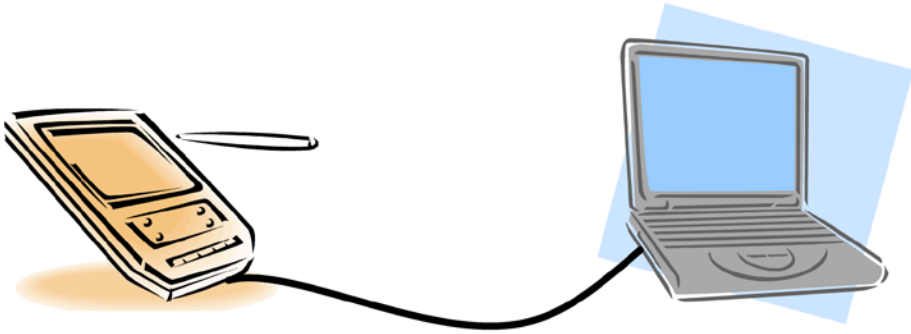

Figure 1-1. Illustration of an Android device in USB device mode

USB Host Mode

When a USB device is connected to an Android device, as illustrated in Figure 1-2, the Android device is said to be in USB host mode, and the Android device has to supply power to the connected device. An Android device functioning as a USB embedded host or as an On-The-Go (OTG) host must supply 5V/500mA of power when the connected device is USB bus powered.

Figure 1-2. Illustration of an Android device in USB host mode

There is also a unique Android USB setup, which was introduced during the Honeycomb version of Android, named the USB accessory mode.

USB Accessory Mode

In USB accessory mode, an Android device that is in the USB device mode can manage external devices. This ability is achieved by connecting the Android device to an external embedded accessory device, which acts as a USB host. The Android device goes to USB accessory mode in order to manage devices that connect to the accessory device. Figure 1-3 depicts Android accessory mode with a simple illustrative example of managing a camera from an Android device using an accessory device. Accessory mode is explained in detail in Chapter 5, which will provide you with a better understanding of the process.

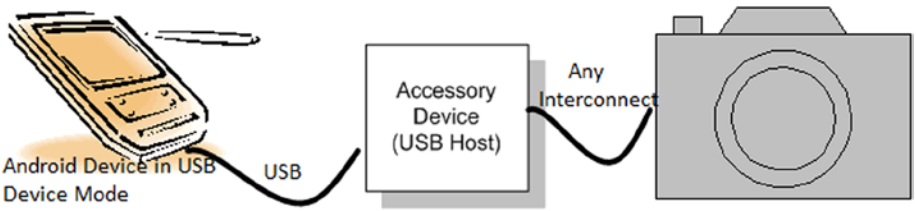

Figure 1-3. *Illustration of an Android device in USB accessory mode*

The USB section of Android CDD defines which USB functionalities have to be supported in the host and device modes. Tables 1-1 and 1-2 capture the requirements when an Android device acts as a USB device or as a USB host.

Table 1-1. *Illustration of an Android CDD 4.4 as Defined in USB Device Requirements*

USB Device Requirement
■ The port must be connectable to a USB host with a standard USB-A port.
■ The port should use the micro-USB form factor on the device side. Existing and new devices that run Android 4.4 are very strongly encouraged to meet these requirements in Android 4.4 so that they will be able to upgrade to future platform releases.
■ The port should be centered in the middle of an edge. Device implementations should either locate the port on the bottom of the device (according to natural orientation) or enable software screen rotation for all apps (including the home screen), so that the display draws correctly when the device is oriented with the port at the bottom. Existing and new devices that run Android 4.4 are very strongly encouraged to meet these requirements in Android 4.4 so that they will be able to upgrade to future platform releases.

(*continued*)

Table 1-1. (*continued*)

USB Device Requirement
■ If the device has other ports (such as a non-USB charging port) it should be on the same edge as the micro-USB port. ■ It must allow a host connected to the device to access the contents of the shared storage volume using either USB Mass Storage Protocol or the Media Transfer Protocol. ■ It must implement the Android Open Accessory API and specification as documented in the Android SDK documentation, and also must declare support for the hardware feature `android.hardware.usb.accessory` [Resources, 52]. ■ It must implement the USB audio class (version not mentioned in CDD) as documented in Android SDK documentation (http://developer.android.com/reference/android/hardware/usb/UsbConstants.html#USB_CLASS_AUDIO). ■ It should implement support for USB battery charging specification (version 1.2) [Resources, 64]. Existing and new devices that run Android 4.4 are very strongly encouraged to meet these requirements in Android 4.4, so that they will be able to upgrade to future platform releases. ■ Device implementations must implement the Android Debug Bridge. If a device implementation omits a USB client port, it must then implement the Android Debug Bridge via a local area network (such as Ethernet or 802.11).

Table 1-2. *Illustration of an Android CDD 4.4 as Defined in USB Host Requirements*

USB Host Requirement
■ It may use a non-standard port form factor, but if so, the device must be shipped with a cable or cables that will adapt the port to a standard USB-A. ■ It must implement the Android USB host API as documented in the Android SDK and declare support for the hardware feature `android.hardware.usb.host` (http://developer.android.com/guide/topics/usb/host.html).

These requirements are defined in section 7.7 USB of the Android CDD 4.4, and you should also note that the requirements are brief and point to the actual specifications. It is important to note that there are few requirements that define actual physical characteristics of an Android device. These physical characteristics will be handy when maintaining compatibility with external accessories, such as audio docks.

Over and above these two tables, USB requirements can also be found across other sections such as "Memory and Storage." The following snippet captures one such requirement from the storage section of CDD:

> *"Regardless of the form of shared storage used, device implementations MUST provide some mechanism to access the contents of shared storage from a host computer, such as USB mass storage (UMS) or Media Transfer Protocol (MTP). Device implementations MAY use USB mass storage, but SHOULD use Media Transfer Protocol. If the device implementation supports Media Transfer Protocol:*
>
> - The device implementation should be compatible with the reference Android MTP host and Android File Transfer [Resources, 57].
> - The device implementation should report a USB device class of `0x00`.
> - The device implementation should report a USB interface name of `MTP`.
>
> *If the device implementation lacks USB ports, it must then provide a host computer with access to the contents of the shared storage by some other means, such as a network file system."*

The storage section defines how the storage space of an Android device should be shared by a host PC over USB. The storage section explains in detail mandating MTP as the preferred USB protocol for sharing the storage space.

DID YOU KNOW?

Have you ever wondered why your Android device is not enumerating as Mass Storage device from Ice Cream Sandwich and later? The secret lies in Android CDD. From the following two snippets, it is very apparent that Android has moved from Mass Storage to MTP as the default mechanism to connect to the host computer.

Android CDD 2.3 – Ginger Bread Version

"It must implement the USB mass storage specification, to allow a host connected to the device to access the contents of the /sdcard volume."

Android CDD 4.0.3 – Ice Cream Sandwich Version

"Regardless of the form of shared storage used, device implementations MUST provide some mechanism to access the contents of shared storage from a host computer, such as USB mass storage (UMS) or Media Transfer Protocol (MTP). Device implementations may use USB mass storage, but should use Media Transfer Protocol."

Now that you are able to understand Google Android's USB requirements, you can now explore how these requirements are built within the Android framework.

Android USB Architecture

This section explains Android USB architecture based on the various USB modes in which an Android device can perform as explained in the initial section. In simple terms, an Android platform is made of Android Linux kernel as the base to manage the platform resources. A Java-based Android framework sits on top of Android Linux kernel, providing the necessary user experience. Some Android features lay within the kernel, and certain features are available only at the Android framework. In case of USB, the functionality is managed between the Android Linux kernel and the user space Android framework.

DID YOU KNOW?

An important point to note is that the kernel discussed here is called the "Android Linux kernel" because it's not same as the generic Linux kernel, and most importantly, not the same as the Linux USB gadget framework. The USB device stack is referred to as the USB gadget framework, and is yet to be integrated as part of the mainline kernel.

The following section provides a top-level architectural view of Android USB in USB device mode, detailing the complete Android USB starting from the Android Linux kernel to the user space Android framework.

When you connect an Android device to a host PC, the Android device is said to be in USB device mode and can export multiple USB functionalities like MTP, ADB, or CDC to the host PC through its descriptors. This type of USB device is referred as a composite device, where a single USB device supports multiple USB functions through their interfaces. From the architecture diagram shown in Figure 1-4, you can infer that the composite infrastructure is part of the kernel and most of the USB device functions are

implemented as “class drivers” within the Android Linux kernel. There are exceptions, like ADB and MTP, which are implemented on both sides, i.e. the kernel and user space. In such cases, the kernel driver implements just the transport part of USB, guaranteeing delivery of the data. The Android framework performs the functional management, implementing the class-level protocol, which other chapters of this book will explore in more detail later. The following section provides a brief overview of the architectural blocks used in the USB device mode, as represented in Figure 1-4.

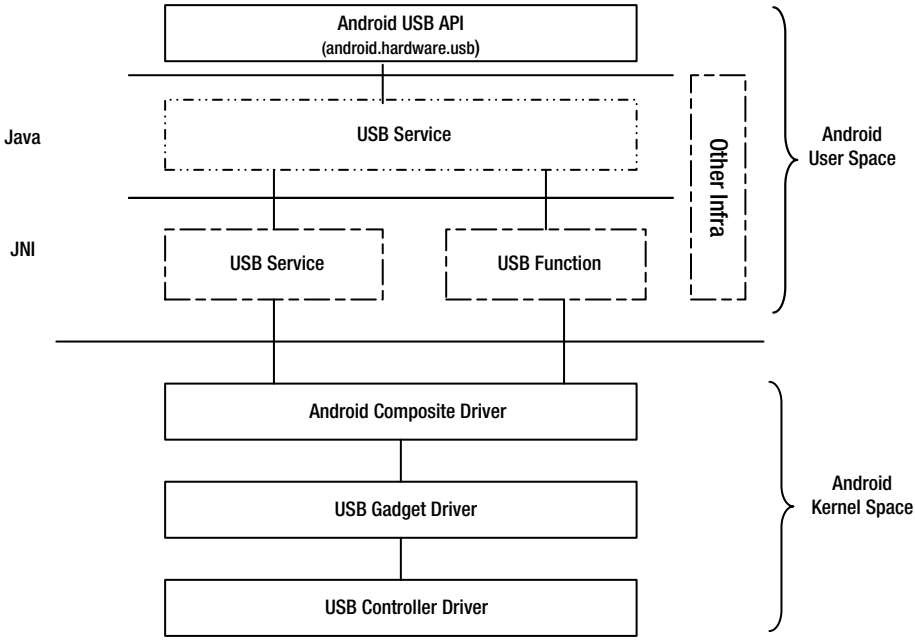

Figure 1-4. Android USB device framework architecture

USB Service

The USB Service framework is the key factor and is the backbone in Android USB device mode. In a way, the role of this framework is to listen to and communicate state changes in Android kernel USB driver and subsequently pass that information on to other interested Android frameworks. Those frameworks then pass that information further to other modules by broadcasting their intent with only the necessary information. This framework also manages USB functions that an Android device has to share when connected to a host PC. More details about this framework will be explained further in Chapter 2, entitled “Discovering and Managing USB within Android.”

USB Function

Most of the USB functions are implemented in the Android Linux kernel space. However, USB functions like ADB or MTP are implemented as user space daemons integrated within the Android framework. This block represents the daemons that implement USB Class requirements. Subsequent chapters on ADB and MTP provide a detailed view on how this module interacts with the kernel below and other Android frameworks.

android.hardware.usb

Android APIs for USBs are represented as a `android.hardware.usb` package and are discussed in further detail in later sections of this chapter. In a USB device mode, these APIs have a minimal role, as there are no APIs that allow managing a USB device's functionality. The exception to this is Android accessory mode, where developers are required to write applications to manage external devices over USB device mode.

Other Infra

Within the Android framework there are many other frameworks that are interested in the USB state changes, like connection, disconnection, or a switch of USB functionalities. This "other infra" represents Android modules like storage infrastructure, network daemon infrastructure, and charging infrastructure, to name a few that are interested in USB state changes. These other infrastructures hook themselves up to the USB framework for the *Intent* that the USB Service module generate. In Chapter 2, we will provide some insight into how to listen to USB states changes. Other chapters will deal with storage and tethering, including details of how they hook and receive the necessary information.

This module also represents the user interface part of Android that communicates USB state changes to the user over the Notification panel. The Android USB architecture is the same in USB accessory mode and USB device mode, as accessory mode is nothing but the USB device mode with some deviation.

Now that you understand how the Android framework in USB device mode works, you can explore the Android framework in USB host mode. Similar to device mode, host mode keeps most of the class functions implemented within the Linux kernel, but classes like `MTP host mode` are implemented in Android user space. It is important to note that, unlike the device stack (gadget driver), which differs from the mainline Linux kernel, the USB host stack is same as the mainline Linux kernel. Though Linux kernel has support for almost all USB devices, an Android device in USB host mode might not