Technical Notes

TN_134

FTDI Android D2XX Driver

Version 1.5

Issue Date: 25-09-2024

This document describes the installation and use of the FTDI D2XX driver for FTxxxx devices in an Android environment.

Future Technology Devices International Limited (FTDI)
Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777   Fax: + 44 (0) 141 429 2758
Web Site: http://ftdichip.com

Table of Contents

1 Introduction

FTDI provides the proprietary D2XX interface for easily communicating with its FTxxxx devices. The D2XX API is common across several operating systems supported by FTDI, namely Windows, Windows CE, Linux, and Mac OS X.

1.1 Android Support

To support the popular Google Android OS, FTDI has prepared three D2XX solutions for different application scenarios.

1. A Java class which uses the JNI (Java Native Interface) to access the API of a pre-compiled Linux D2XX library. This solution is applicable to all versions of Android platforms but requires special root privilege on USB related device nodes. This interface is useful when you want to reuse existing design for Linux in Android applications.
2. A Java class built on top of the emerging USB Host support available since Android version 3.1. This set of libraries is applicable to Android v3.1 or latter system and requires no special root access privilege as previous solution do. As this requires no special root access privileges, FTDI recommends it for all applications on newer platforms.
3. A precompiled Linux D2XX library which is built by Android Open-Source Project (AOSP). This library is applicable to Android to v9.0.8 or latter system and requires special root privileges on USB related device node.

The library file for each solution is different by its nature. For the first solution, the library provided a compiled native library (libftd2xx-jni.so), a D2XX Java class file (D2xx.java) as an interface for Android application; the source to the JNI portion of the native library is also available and is statically linked with the native libftd2xx.a library to produce the Java compatible libftd2xx-jni.so file. For the second solution, there is a library called d2xx.jar that provides end-users an easy way to integrate D2XX APIs. For the third solution, a library called libftd2xx.so needs to be built along with the Android open source. This library can be used by other Android service or 3rd party AOSP library.

We strongly recommend using the 2nd solution to implement an Android application. Examples of solution #2 can be downloaded from the D2XX Drivers page on the FTDI website here - https://ftdichip.com/drivers/d2xx-drivers/. After extracting the archive file, the folder *TN_147* contains a subdirectory called *javad2xxdemo* which contains files for solution #2. Folder *AN_357* contains another subdirectory called *javad2xxdemo* which contains files for solution #2.

1.2 Prerequisites

To install the FTDI D2XX driver and test it successfully, the following are required:

- For the first solution, a hardware platform including a USB host device supported by the Android/Linux kernel.
  - FTDI testing was conducted using a BeagleBoard-xM Rev C.
- For the second solution, an Android device/BSP supporting Android USB Host API is also required,
  - FTDI suggests using a BSP corresponding to AOSP 4.0 or latter.
  - If such a BSP is not available, any contemporary Android devices running v3.1 or latter OS, with USB Host or OTG interface will do. FTDI testing was conducted using a Google Nexus 7.
- An FTDI based device for testing with
  - FTDI testing was conducted with an FT232R based US232R cable.

In addition, to develop an application using the FTDI D2XX driver for Android, a development machine must have the Eclipse IDE and up-to-date Android SDK, including the ADB program and Android ADT Plugin installed. The installation and configuration of these tools is not within the scope of this document and is outlined on the Android developer web site (https://developer.android.com/studio).

The Android device should also have USB Debugging enabled to allow access using the ADB utility. To accomplish this, navigate to Settings > Applications > Development and check the USB debugging option.

A summary of the required configuration is provided in the diagram below.

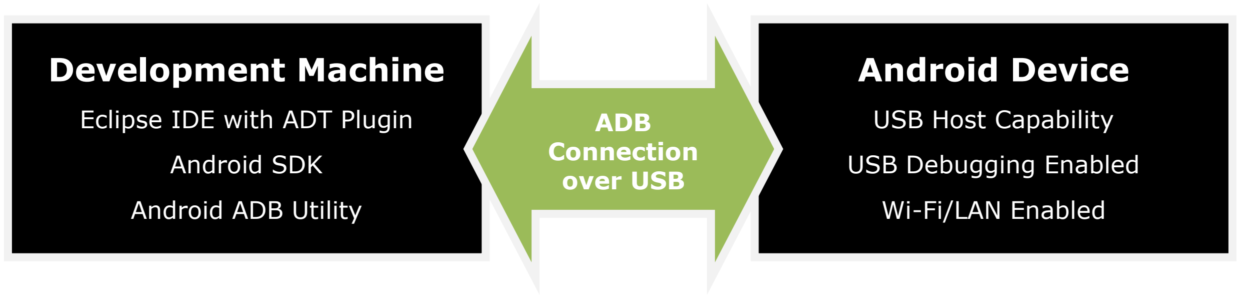


Figure 1 - Android Development Configuration

2 Using the FTDI Native Linux D2XX Library in Android

2.1 Introduction and Usage

To accompany the native D2XX library, FTDI have provided a Java class and a JNI wrapper which can be easily included in an application. The class provides access to all the classic D2XX functions including EEPROM functions. The D2xx Java class can readily be included in an Android application project in Eclipse.

The D2xx class (not to be confused with the D2XX native API) provides some static methods that allow access to driver-wide information such as the VID and PID combinations to match with and the device information list.

All other methods require a D2xx object to be created and subsequently opened using one of the four open methods (openByIndex, openBySerialNumber, openByDescription, or openByLocation). Executing an open method (if successful) will cause the instance of the D2xx class to internally maintain a native handle value; this is used for all subsequent communication with the device. When the device is no longer required, the native handle can be closed with the close method.

Since the JNI calls ultimately call native D2XX functions, exception generation is included in the JNI layer. An exception of type D2xxException (extended from IOException) is thrown in the case of a native D2XX call returning an FT_STATUS code other than FT_OK. The exception also generates a message indicating the native status code and the native function that the exception occurred in.

The D2xx class is fully documented using the Javadoc standard. For information on the D2xx class methods, constants, and sub-classes, please consult the Javadoc entry for the item of interest.

A sample application demonstrating how to use various methods in the D2xx class is also provided to assist custom application development. The sample application is shown below:

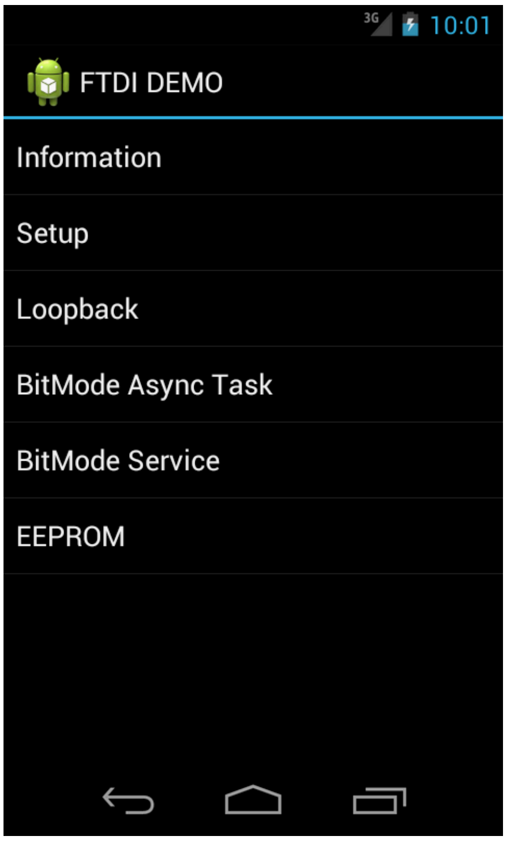
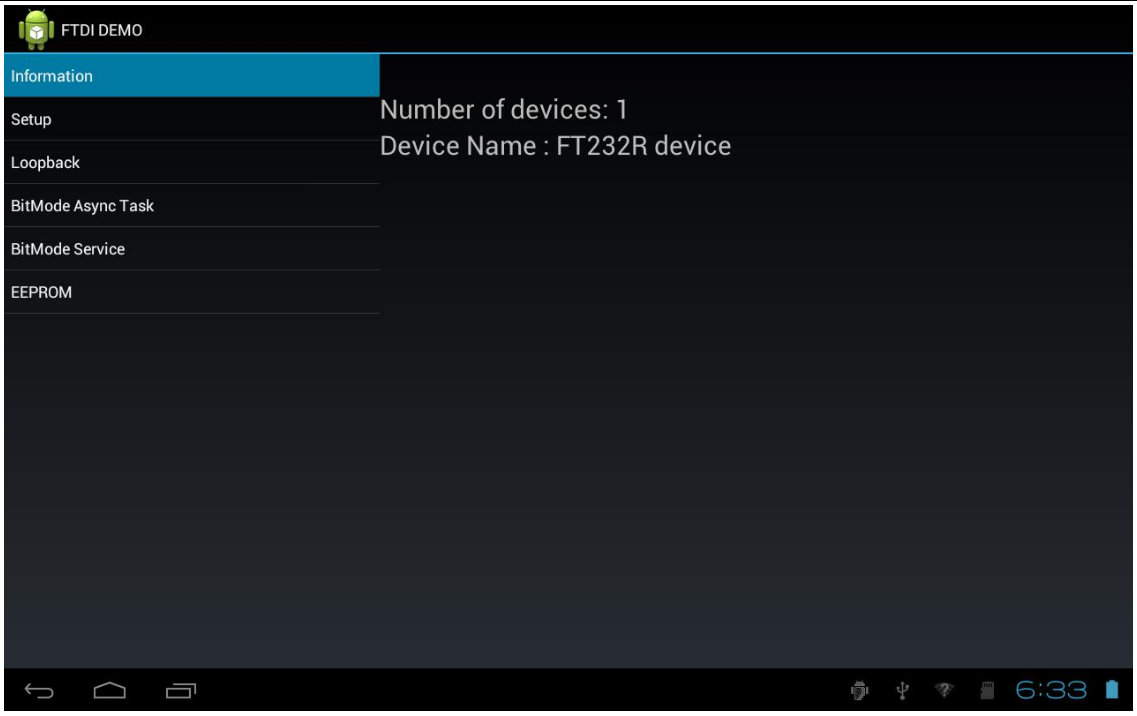


Figure 2 - D2XX Demo Application running on an Android phone



Number of devices: 1
Device Name : FT232R device

Figure 3 - D2XX Demo Application running on an Android tablet

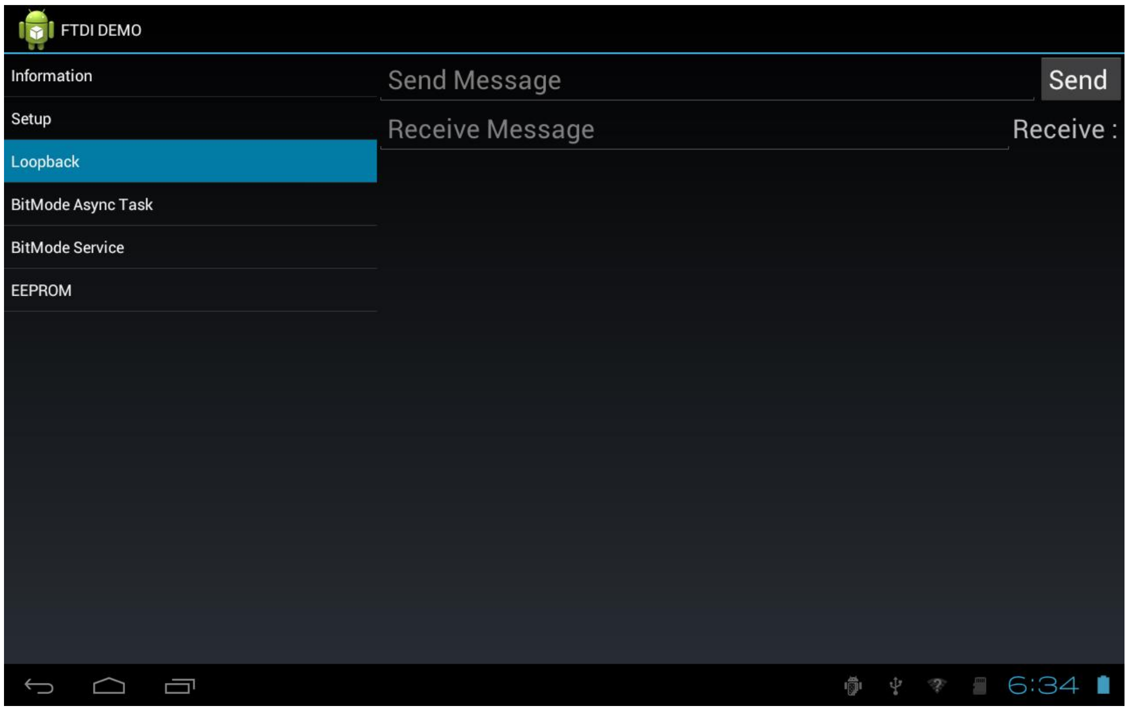


Figure 4 - D2XX Loopback test running on a tablet

Tapping the Information item in the sample application will display the number of devices available and the chip type of the first device in the device list.

Tapping the Loopback item and writes some text on the “Send Message” edit box, then tap “Send” button, will write the message to the device; if some text is toasted back, then it will show up in the “Receive Message” area. In the above screen shot, a loopback connector was fitted so the data received will be the same as the data sent.

2.2 Limitations, Restrictions and Reference

2.2.1 Non-Default VID and PID Combinations

At the time of writing, the Android D2XX driver will support all default FTDI VID and PID combinations and can also support a single custom VID and PID combination via the setVIDPID Java method (FT_SetVIDPID function).

However, it is currently not possible to match several non-default VID and PID combinations simultaneously. This is due to the Android OS hanging on a call to dlopen which precludes the use of an external libtable library at this time.

2.2.2 USB Device Permissions

Many Android systems with USB host capability enumerate devices with application incompatible permissions (0660). These permissions are specified in the ueventd.rc file as follows:

```
/dev/bus/usb/*    0660  root  usb
```

To modify the default permissions for USB devices in the ueventd.rc file a user must have root access. This may render the D2XX library unusable on devices that have default permissions of 0660 and do not allow root access.

To make the device accessible, the ueventd.rc entry listed above should be changed to make the device world readable and world writeable as follows:

```
/dev/bus/usb/*    0666  root  usb
```

NOTE: FTDI does not accept any responsibility for customers who choose to enable root access on their Android platform and subsequently damage the unit or void the warranty. Enabling root access on such devices is entirely at the user's own risk.

2.2.3 Reference

The source code for JNI method is available here:
https://www.ftdichip.com/old2020/Drivers/D2XX/D2XXSample/D2XXSample.zip

This link and method are no longer supported. If you have specific requirements to use this method, please contact your local FTDI support team. Otherwise, we strongly recommend using J2XX to implement the JAVA application.

3 Using the Java FTDI D2XX Library in Android

3.1 Introduction and Usage

To support versatile tablet usage scenario, Google has added USB Host API to Android since version 3.1. Before this very version, an Android application cannot access USB devices attached to system naturally without root access right. The Android USB Host API breaks through the limitation, now we can utilize USB gadgets attached to Android Host or OTG port without further ado.

But communicating via raw USB data to gadgets is not easy for developer. FTDI have provided a Java class library which can be easily adapted in applications, developer can focus on desired I/O read and write, without caring complex USB device setup. The goal of class library design is to provide access to all the classic D2XX functions including EEPROM functions, but at the time of writing this document, the D2XX library is still in alpha phase, thus offer limited functionality. Available API is listed in the limitation section.

The D2xx Java library can be readily included in an Android application project in Eclipse or Android Studio. The D2xx class (not to be confused with the previous D2XX interface to native API) provides some static methods that allow access to driver-wide information such as the VID and PID combinations to match with and the device information list.

All other methods require a D2xx object to be created and subsequently opened using one of the four open methods (openByIndex, openBySerialNumber, openByDescription, or openByLocation). Executing an open method, if successful, will cause the instance of the D2xx class to internally maintain a native handle value; this is used for all subsequent communication with the device. When the device is no longer required, the native handle can be closed with the close method.

The D2xx class is fully documented using the Javadoc standard. For information on the D2xx class methods, constants, and sub-classes, please consult the Javadoc entry for constants, of interest.

A sample application demonstrating how to use various methods in the D2xx class is also provided to assist custom application development. The sample application is shown below:

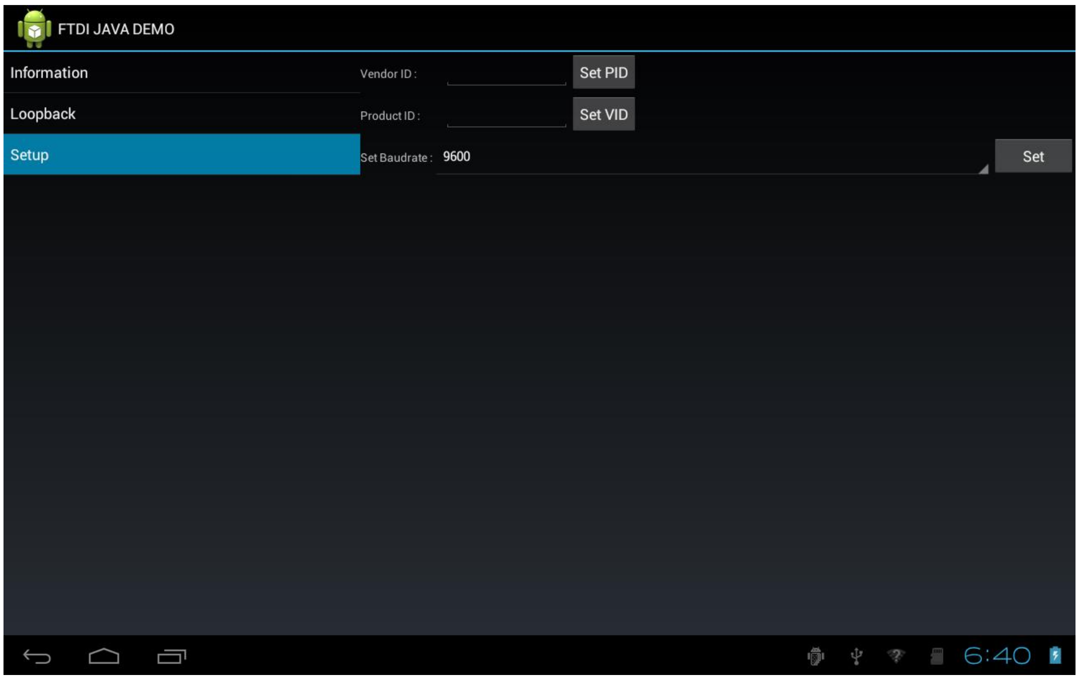
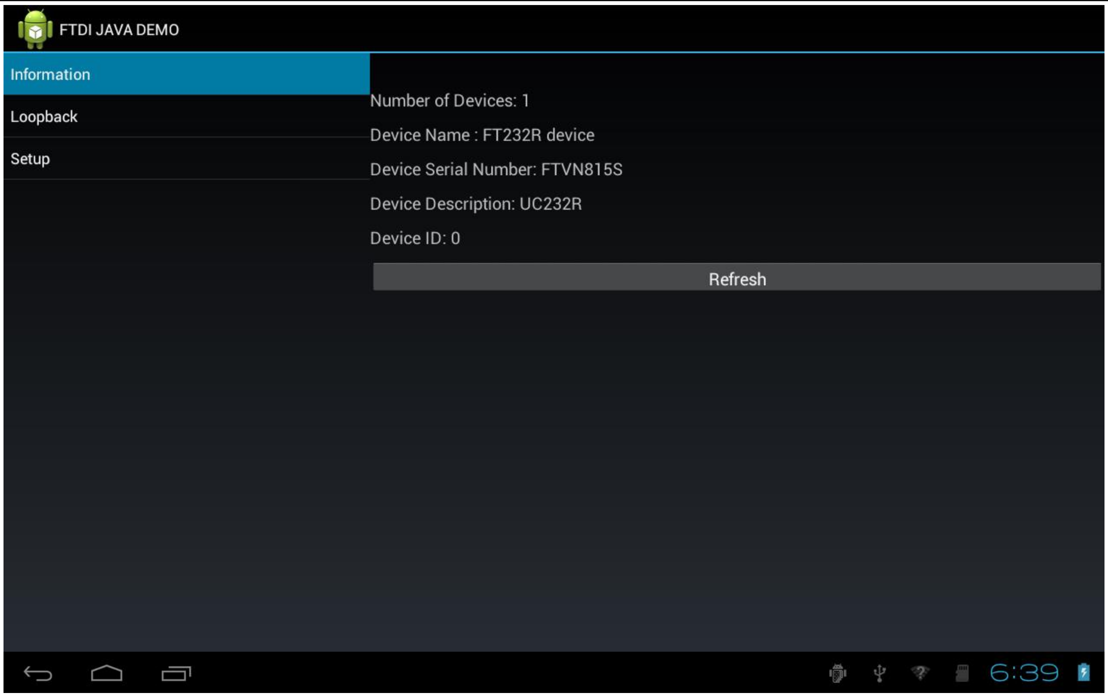


Figure 5 - D2XX Demo Application using the new API running on an Android tablet


FTDI JAVA DEMO

Information
Loopback
Setup

Number of Devices: 1
Device Name : FT232R device
Device Serial Number: FTVN815S
Device Description: UC232R
Device ID: 0

Figure 6 - D2XX Demo Application using the new API to get device information

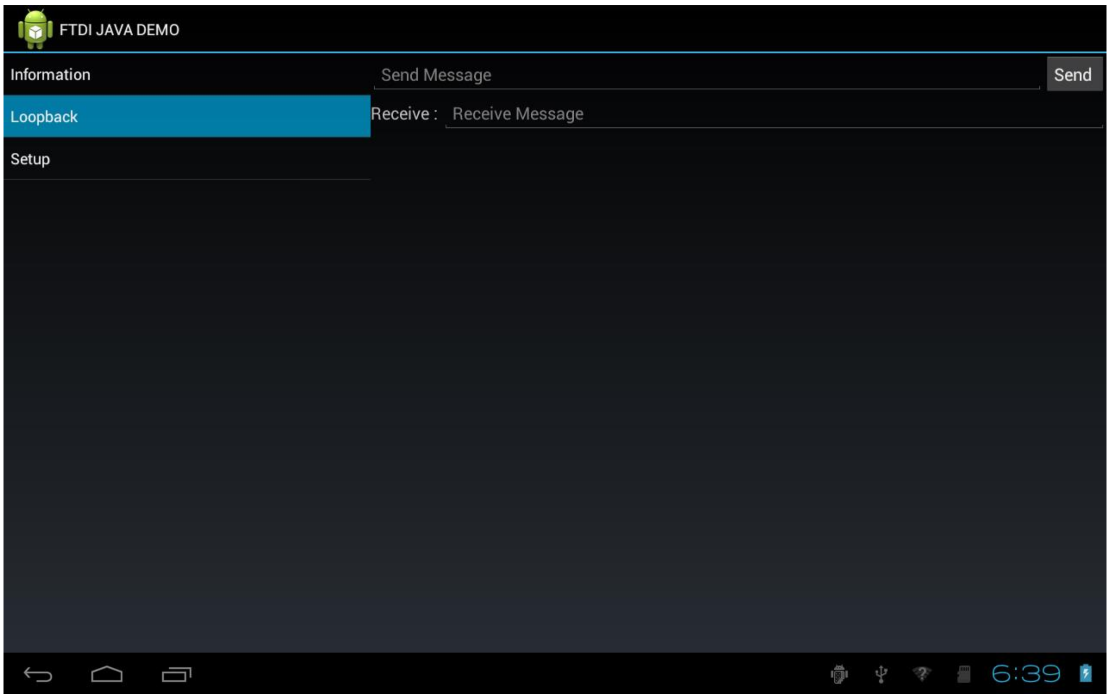


Figure 7 - D2XX Loopback test using the new API

Tapping the Information item in the sample application will displays the number of devices available and the chip type of the first device in the device list.

Tapping the Loopback item and writes some text on the “Send Message” edit box, then tap “Send” button, will write the message to the device; if some text is toasted back, then it will show up in the “Receive Message” area. In the above screen shot, a loopback connector was fitted so the data received will be the same as the data sent.

3.2 Limitations and Restrictions

3.2.1 Non-Default VID and PID Combinations

At the time of writing, the Android D2XX driver will support all default FTDI VID and PID combinations and can also support a single custom VID and PID combination via the setVIDPID Java method (FT_SetVIDPID function).

However, it is currently not possible to match several non-default VID and PID combinations simultaneously. This is due to the Android OS hanging on a call to dlopen which precludes the use of an external libtable library at this time.

3.2.2 Currently Available D2XX Functions

As this solution is still in alpha phase, only part of classical D2XX API is supported, here is a table of useable API in this release:

API	Description
setVIDPID	Set a specific combination of VID/PID pair to use
createDeviceInfoList	This method builds an internal device information list and returns the number of D2XX devices connected to the system.
getDeviceInfoList	This method returns the device list created with a prior call to createDeviceInfoList.
getDeviceInfoListDetail	This method returns information for a single device from the internal device list.
openByIndex	This method opens the device at the specified index for use.
openBySerialNumber	This method opens the device with the specified serial number for use.
openByDescription	This method opens the device with the specified description for use.
openByLocation	This method opens the device at the specified location for use.
isOpen	Returns the open status of the device.
close	This method closes an opened device.
read	This method reads data from the device into the Java application buffer.
write	This method writes data to the device from the Java application buffer.
setBaudRate	This method sends a vendor command to the device to change the baud rate generator value.
setDataCharacteristics	This method dictates the data format that the device will use.
setFlowControl	This method specifies the flow control method that the device should use.
setDtr	This method allows the DTR modem control line to be manually asserted.

API	Description
clrDtr	This method allows the DTR modem control line to be manually de-asserted.
setRts	This method allows the RTS modem control line to be manually asserted.
clrRts	This method allows the RTS modem control line to be manually de-asserted.
getQueueStatus	This method retrieves the number of bytes available to read from the native driver Rx buffer.
purge	Discards any data form the specified driver buffer and flushes data from the device.
resetDevice	This method sends vendor commands to the device to cause a reset and flush any data from the device buffers.
getDeviceInfo	This method retrieves information on the device that is currently open.
setLatencyTimer	This method allows the latency timer value for the device to be specified.
getLatencyTimer	This method retrieves the latency timer value from the specified device.
setBitMode	This method allows the device to use alternative interface modes such as bit-bang, MPSSE and CPU target mode.

Table 1 – API and Description

3.2.3 Reference

This lib is already published to FTDI website.
The link is https://ftdichip.com/software-examples/android-java-d2xx/
We strongly recommend using this lib to implement Android application development.

4 Using the Android AOSP D2XX Library

4.1 Introduction and Usage

The Android Open System Platform (AOSP) is publicly available and modifiable Android source code. Anyone can download and modify AOSP for their device. AOSP provides a complete and fully functional implementation of the Android mobile platform. The link library files, or executable files generated in AOSP Build System are the library files that the Android system must rely on to run.

The AOSP library is designed for other library or application running in C Framework. If you want to gather data in the background and the gathered data may not be returned to user immediately. Notice, this lib needs root privileges to run.

4.2 Differences between Android NDK and AOSP Build System

The developer group that Android NDK faces is APP developers. They want to use C/C++ code to achieve certain functions, and then call these functions through JNI in the Java code of the upper APP. For example, developers of some mobile game apps, to make the screen smoother when the game is running, they often use the C/C++ code to call the OpenGLES API function to implement these image rendering modules, which are time-consuming and require high performance. Then it is compiled into a shared library file through ndk-build, and then loaded and called by the Java code in the upper APP.

AOSP Build System is aimed at developers who are developers of the underlying operating system. They need to make a custom modification to the Android source code according to their own needs and the characteristics of the hardware platform, and then recompile through the AOSP Build System to get the Android System images and library files they want.

4.2.1 Reference

AOSP lib is not published to FTDI website yet. If you want to know more about this library, please contact your local FTDI team.

5 The relationship on these three libraries

5.1 The user scenarios for these three FTDI libraries

Using the FTDI Native Linux D2XX Library:

- The library is designed for Java applications which can through JNI interface to access it.
- The library requires root privilege.
- This library may have higher performance.
- Now, only armv8 platform is available for this library.

Using the New Java FTDI Android D2XX Library

- The library is designed for Java applications which is built by java.
- The library does not require root privilege.
- The application is easy to move to another android platform.

Using the Android AOSP D2XX Library

- The library is designed for other library or application running in C Framework.
- This library needs to fit the AOSP version. When Android versions higher or equal to 9.0, Android.bp configuration is used. Otherwise, Android.mk is used.
- The library requires root privileges.

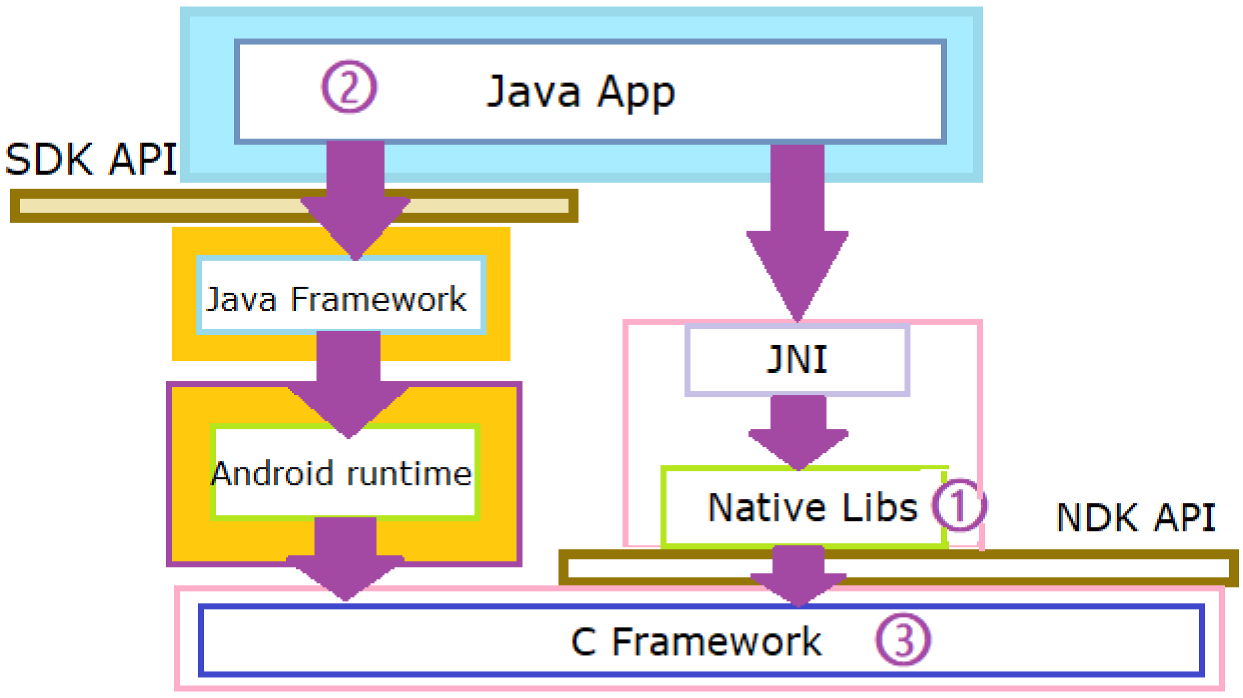


Figure 8 – Three D2xx libraries in Android

6 Contact Information

Head Office – Glasgow, UK
Future Technology Devices International Limited (UK)
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales)	sales1@ftdichip.com
E-mail (Support)	support1@ftdichip.com
E-mail (General Enquiries)	admin1@ftdichip.com

Branch Office – Tigard, Oregon, USA
Future Technology Devices International Limited (USA)
7130 SW Fir Loop
Tigard, OR 97223-8160
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales)	us.sales@ftdichip.com
E-Mail (Support)	us.support@ftdichip.com
E-Mail (General Enquiries)	us.admin@ftdichip.com

Branch Office – Taipei, Taiwan
Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan, R.O.C.
Tel: +886 (0) 2 8797 1330
Fax: +886 (0) 2 8751 9737

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com

Branch Office – Shanghai, China
Future Technology Devices International Limited (China)
Room 1103, No. 666 West Huaihai Road,
Shanghai, 200052
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales)	cn.sales@ftdichip.com
E-mail (Support)	cn.support@ftdichip.com
E-mail (General Enquiries)	cn.admin@ftdichip.com

Web Site
http://ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A - References

Document References

NA

Other References

http://developer.android.com

http://code.google.com/p/rowboat/wiki/JellybeanOnBeagleboard_WithSGX

http://beagleboard.org/hardware-xM

Acronyms and Abbreviations

Terms	Description
ADT	Android Development Tools
API	Application Programming Interface
EEPROM	Electrically Erasable Programmable Read-Only Memory
FTDI	Future Technology Devices International
JNI	Java Native Interface
OS	Operating System
SDK	Software Development Kit
USB	Universal Serial Bus

Appendix B – List of Tables and Figures

List of Tables

List of Figures

Appendix C – Revision History

Document Title:	TN_134 FTDI Android D2XX Driver
Document Reference No.:	FT_000522
Clearance No.:	FTDI#223
Product Page:	https://ftdichip.com/product-category/products/
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial Release for beta test.	2011-09-29
1.1	Modifications file path of adb psuh in section 2.	2012-08-16
1.2	Modifications for new sample application.	2012-09-17
1.3	Corrected the broken link to driver; Updated Contact Info details and Copyright information.	2017-06-08
1.4	Added AOSP description.	12-10-2023
1.5	Fixed some description errors.	25-09-2024