Future Technology Devices International Ltd.

Java D2xx for Android API User Manual

Document Reference No.:FT_000796

Version 1.0

Issue Date: 2013-02-05

This document provides the application programming interface (API) for the Java D2xx for Android library.

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
E-Mail (Support): support1@ftdichip.com **Web:** http://www.ftdichip.com

Table of Contents

1 Preface

The D2xx interface is a proprietary interface specifically for FTDI devices. This document provides an explanation of the functions available to application developers via the D2xx library.

The software code examples used in the examples in this manual are not guaranteed nor are they supported by FTDI.

1.1 Acronyms and Abbreviations

Terms	Description
D2xx	FTDI's proprietary "direct" user space driver interface running on-top of Android USB Host API
OS	Operating System
USB	Universal Serial Bus
BSP	Board Supporting Package
WORD	16 bits data
Break	A signal in the UART protocol
API	Application Programming Interface
OTG	On The Go
SDK	Software Development Kit
ADT	Android Development Tools
IDE	Integrated Development Environment
ADB	Android Debug Bridge
EEPROM	Electrically Erasable Programmable Read Only Memory
CBUS	CBUS GPIO Pin
WiFi	Wireless Fidelity
LAN	Local Area Network
MCU	Microcontroller Unit
SYNC	Synchronous
ASYNC	Asynchronous
MPSSE	Multi-Protocol Synchronous Serial Engine
FIFO	First In First Out
CTS	Clear To Send
RTS	Request To Send

2 Introduction

FTDI provides a proprietary Android D2xx library for easy communication with its FTxxxx devices. The D2xx API is an Android operating system library supported by FTDI.

2.1 Android Support

The API listed in this document is a D2xx solution to application scenarios supporting the Google Android OS.

A Java class library supporting USB Host is available and applicable to Android v3.2 or any later series. This library requires no special root access privileges.

2.2 Prerequisites

The following is required to install the FTDI D2xx driver:

- An Android device(recommended),
    - A BSP supporting Android USB Host API corresponding to AOSP 3.2 or later
    - A contemporary Android device running v3.2 or a later OS, with USB Host or OTG interface. FTDI testing was conducted using a Google Nexus 7.
- An FTDI chip based module to test the FTDI D2xx driver:

NOTE:

To develop an application using the FTDI D2xx driver for Android, the development machine must have the Eclipse IDE and an up-to-date version of Android SDK, including the ADB program and Android ADT Plugin installed. The installation and configuration of these tools is not included in this document. For more information, please see (http://developer.Android.com/sdk/index.html).

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.

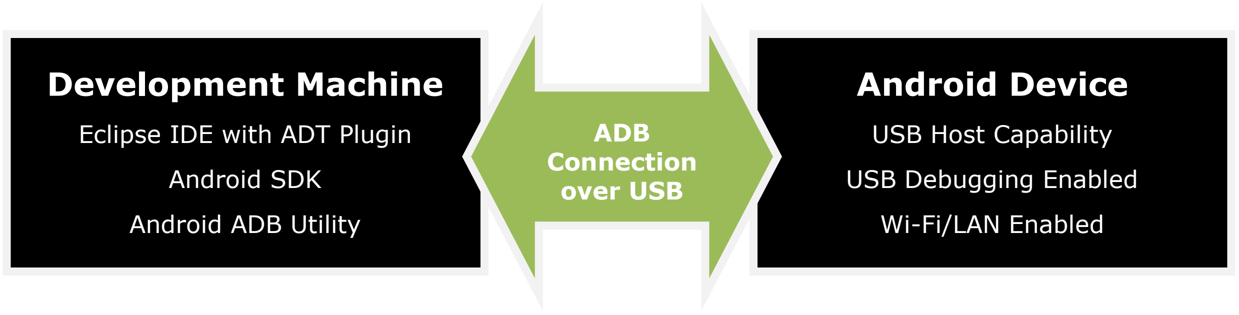

Figure 1: Android Development Configuration

3 D2xx Library Packages

3.1 Package com.ftdi.j2xx

Class Summary

Class	Description
D2xxManager	A management class for connected FTDI devices.
D2xxManager.DriverParameters	A class for read() parameters.
D2xxManager.FtDeviceInfoListNode	Information about a connected FTDI device.
FT_Device	A device class providing different APIs for a host to communicate and operate different FT devices.
FT_EEPROM	EEPROM data structure of the 232A, 232B
FT_EEPROM_2232D	EEPROM data structure on the 2232D
FT_EEPROM_2232H	EEPROM data structure on the 2232H
FT_EEPROM_2232H.DRIVE_STRENGTH	The driver strength of the 2232H
FT_EEPROM_232H	EEPROM data structure on the 232H
FT_EEPROM_232H.CBUS	CBus Option of the FT232H
FT_EEPROM_232H.DRIVE_STRENGTH	The driver strength on the 232H
FT_EEPROM_232R	EEPROM data structure on the 232R
FT_EEPROM_232R.CBUS	CBus Option on the FT232R
FT_EEPROM_245R	EEPROM data structure on the 245R
FT_EEPROM_245R.CBUS	CBus Option on the FT245H
FT_EEPROM_4232H	EEPROM data structure on the 4232H
FT_EEPROM_4232H.DRIVE_STRENGTH	The driver strength on the FT4232H
FT_EEPROM_X_Series	EEPROM data structure on the X Series
FT_EEPROM_X_Series.CBUS	CBus Option on the X Series
FT_EEPROM_X_Series.DRIVE_STRENGTH	The driver strength on the X Series

Exception Summary

Exception	Description
D2xxManager.D2xxException	A class for exception debug Handle exception and print error message

3.2 Class Hierarchy

- java.lang.Object
  - com.ftdi.j2xx.**D2xxManager**
  - com.ftdi.j2xx.**D2xxManager.DriverParameters**
  - com.ftdi.j2xx.**D2xxManager.FtDeviceInfoListNode**
  - com.ftdi.j2xx.**FT_Device**
  - com.ftdi.j2xx.**FT_EEPROM**
    - com.ftdi.j2xx.**FT_EEPROM_2232D**
    - com.ftdi.j2xx.**FT_EEPROM_2232H**
    - com.ftdi.j2xx.**FT_EEPROM_232H**
    - com.ftdi.j2xx.**FT_EEPROM_232R**
    - com.ftdi.j2xx.**FT_EEPROM_245R**
    - com.ftdi.j2xx.**FT_EEPROM_4232H**
    - com.ftdi.j2xx.**FT_EEPROM_X_Series**
  - com.ftdi.j2xx.**FT_EEPROM_2232H.DRIVE_STRENGTH**
  - com.ftdi.j2xx.**FT_EEPROM_232H.CBUS**
  - com.ftdi.j2xx.**FT_EEPROM_232H.DRIVE_STRENGTH**
  - com.ftdi.j2xx.**FT_EEPROM_232R.CBUS**
  - com.ftdi.j2xx.**FT_EEPROM_245R.CBUS**
  - com.ftdi.j2xx.**FT_EEPROM_4232H.DRIVE_STRENGTH**
  - com.ftdi.j2xx.**FT_EEPROM_X_Series.CBUS**
  - com.ftdi.j2xx.**FT_EEPROM_X_Series.DRIVE_STRENGTH**
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - java.io.IOException
        - com.ftdi.j2xx.**D2xxManager.D2xxException**

4 Methods on D2xxManager

A management class for connected FTDI devices. Use "*getInstance()*" to get a copy of D2xxManager; use "createDeviceInfoList()" method to scan current connected FTDI devices, then open target device via a suitable open API.

The functions listed in this section are used to manage FT devices.

Field Summary

Fields

Modifier and Type	Field and Description
static byte	FT_BI Line status bits : OE: FT_OE, PE: FT_PE, FE: FT_FE, BI: FT_BI
static byte	FT_BITMODE_ASYNC_BITBANG Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_CBUS_BITBANG Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_FAST_SERIAL Bit Mode bits : Reset: FT_BITMODE_RESET,

	Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_MCU_HOST Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_MPSSE Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_RESET Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) :

	FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_SYNC_BITBANG Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_BITMODE_SYNC_FIFO Bit Mode bits : Reset: FT_BITMODE_RESET, Asynchronous Bit Bang: FT_BITMODE_ASYNC_BITBANG, MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MPSSE, Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_SYNC_BITBANG, MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_MCU_HOST, Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) : FT_BITMODE_FAST_SERIAL, CBUS Bit Bang Mode (FT232R and FT232H devices only) : FT_BITMODE_CBUS_BITBANG, Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) : FT_BITMODE_SYNC_FIFO
static byte	FT_CTS Modem status bits : CTS: FT_CTS, DSR: FT_DSR, RI: FT_RI, DCD: FT_DCD
static byte	FT_DATA_BITS_7 Data bits : 7 : FT_DATA_BITS_7, 8 : FT_DATA_BITS_8
static byte	FT_DATA_BITS_8 Data bits : 7 : FT_DATA_BITS_7, 8 : FT_DATA_BITS_8
static byte	FT_DCD Modem status bits : CTS: FT_CTS, DSR: FT_DSR,

	RI: FT_RI, DCD: FT_DCD

static int	FT_DEVICE_2232
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_2232H
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_232B
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_232H
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,

	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_232R
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_245R
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_4232H
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_8U232AM
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,

	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_UNKNOWN
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static int	FT_DEVICE_X_SERIES
	Device Type :
	bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0: FT_DEVICE_232B,
	bvdDevice = 0x0200 and iSerialNumber != 0 : FT_DEVICE_8U232AM,
	Unknown : FT_DEVICE_UNKNOWN,
	bvdDevice = 0x0500 : FT_DEVICE_2232,
	bvdDevice = 0x0600 : FT_DEVICE_232R,
	bvdDevice = 0x0600 : FT_DEVICE_245R,
	bvdDevice = 0x0700 : FT_DEVICE_2232H,
	bvdDevice = 0x0800 : FT_DEVICE_4232H,
	bvdDevice = 0x0900 : FT_DEVICE_232H,
	bvdDevice = 0x1000 : FT_DEVICE_X_SERIES

static byte	FT_DSR Modem status bits : CTS: FT_CTS, DSR: FT_DSR, RI: FT_RI, DCD: FT_DCD
static byte	FT_EVENT_LINE_STATUS Event bits : Rx Char Event: FT_EVENT_RXCHAR, Modem Status Event: FT_EVENT_MODEM_STATUS, Line Status Event: FT_EVENT_LINE_STATUS, Removed Event: FT_EVENT_REMOVED
static byte	FT_EVENT_MODEM_STATUS Event bits : Rx Char Event: FT_EVENT_RXCHAR, Modem Status Event: FT_EVENT_MODEM_STATUS, Line Status Event: FT_EVENT_LINE_STATUS, Removed Event: FT_EVENT_REMOVED
static byte	FT_EVENT_REMOVED Event bits : Rx Char Event: FT_EVENT_RXCHAR, Modem Status Event: FT_EVENT_MODEM_STATUS,

	Line Status Event: FT_EVENT_LINE_STATUS, Removed Event: FT_EVENT_REMOVED
static byte	FT_EVENT_RXCHAR Event bits : Rx Char Event: FT_EVENT_RXCHAR, Modem Status Event: FT_EVENT_MODEM_STATUS, Line Status Event: FT_EVENT_LINE_STATUS, Removed Event: FT_EVENT_REMOVED
static byte	FT_FE Line status bits : OE: FT_OE, PE: FT_PE, FE: FT_FE, BI: FT_BI
static byte	FT_FLAGS_HI_SPEED Device info list flags : Device Open Flag: FT_FLAGS_OPENED, Device Hi Speed Flag: FT_FLAGS_HI_SPEED
static byte	FT_FLAGS_OPENED Device info list flags : Device Open Flag: FT_FLAGS_OPENED, Device Hi Speed Flag: FT_FLAGS_HI_SPEED
static short	FT_FLOW_DTR_DSR Flow Control bits : None: FT_FLOW_NONE, CTS/RTS: FT_FLOW_RTS_CTS, DTR/DSR: FT_FLOW_DTR_DSR, XON/XOFF: FT_FLOW_XON_XOFF
static short	FT_FLOW_NONE Flow Control bits : None: FT_FLOW_NONE, CTS/RTS: FT_FLOW_RTS_CTS, DTR/DSR: FT_FLOW_DTR_DSR, XON/XOFF: FT_FLOW_XON_XOFF
static short	FT_FLOW_RTS_CTS Flow Control bits : None: FT_FLOW_NONE, CTS/RTS: FT_FLOW_RTS_CTS, DTR/DSR: FT_FLOW_DTR_DSR, XON/XOFF: FT_FLOW_XON_XOFF
static short	FT_FLOW_XON_XOFF Flow Control bits : None: FT_FLOW_NONE, CTS/RTS: FT_FLOW_RTS_CTS, DTR/DSR: FT_FLOW_DTR_DSR, XON/XOFF: FT_FLOW_XON_XOFF

static byte	FT_OE Line status bits : OE: FT_OE, PE: FT_PE, FE: FT_FE, BI: FT_BI
static byte	FT_PARITY_EVEN Parity bits, used by App : None: FT_PARITY_NONE, Odd: FT_PARITY_ODD, Even: FT_PARITY_EVEN, Mark: FT_PARITY_MARK, Space: FT_PARITY_SPACE
static byte	FT_PARITY_MARK Parity bits, used by App : None: FT_PARITY_NONE, Odd: FT_PARITY_ODD, Even: FT_PARITY_EVEN, Mark: FT_PARITY_MARK, Space: FT_PARITY_SPACE
static byte	FT_PARITY_NONE Parity bits, used by App : None: FT_PARITY_NONE, Odd: FT_PARITY_ODD, Even: FT_PARITY_EVEN, Mark: FT_PARITY_MARK, Space: FT_PARITY_SPACE
static byte	FT_PARITY_ODD Parity bits, used by App : None: FT_PARITY_NONE, Odd: FT_PARITY_ODD, Even: FT_PARITY_EVEN, Mark: FT_PARITY_MARK, Space: FT_PARITY_SPACE
static byte	FT_PARITY_SPACE Parity bits, used by App : None: FT_PARITY_NONE, Odd: FT_PARITY_ODD, Even: FT_PARITY_EVEN, Mark: FT_PARITY_MARK, Space: FT_PARITY_SPACE
static byte	FT_PE Line status bits : OE: FT_OE, PE: FT_PE, FE: FT_FE, BI: FT_BI

static byte	FT_PURGE_RX Purge flags : RX purge flag: FT_PURGE_RX, TX purge flag: FT_PURGE_TX
static byte	FT_PURGE_TX Purge flags : RX purge flag: FT_PURGE_RX, TX purge flag: FT_PURGE_TX
static byte	FT_RI Modem status bits : CTS: FT_CTS, DSR: FT_DSR, RI: FT_RI, DCD: FT_DCD
static byte	FT_STOP_BITS_1 Stop bits : 1: FT_STOP_BITS_1, 2: FT_STOP_BITS_2,
static byte	FT_STOP_BITS_2 Stop bits : 1: FT_STOP_BITS_1, 2: FT_STOP_BITS_2,
static int	FTDI_BREAK_OFF BREAK on is bit 14 in wValue parameter of FTDI_SET_DATA request : UART break on condition: FTDI_BREAK_OFF, UART break off condition: FTDI_BREAK_ON
static int	FTDI_BREAK_ON BREAK on is bit 14 in wValue parameter of FTDI_SET_DATA request : UART break on condition: FTDI_BREAK_OFF, UART break off condition: FTDI_BREAK_ON

Method Summary

Methods

Modifier and Type	Method and Description
int	**addUsbDevice**(UsbDevice dev) This method analyze the dev passed-in, if it's a FTDI device, add it to manageable device list
int	**createDeviceInfoList**(Context parentContext) This method builds an internal device information list and returns the number of D2XX devices connected to the system.
int	**getDeviceInfoList**(int numDevs, **D2xxManager.FtDeviceInfoListNode**[] deviceList) This method returns the device list created with a prior call to **createDeviceInfoList(Context)**.
D2xxManager.FtDeviceInfoListNode	**getDeviceInfoListDetail**(int index) This method returns information for a single device from the internal device list created by a previous call to **createDeviceInfoList(Context)**.

static **D2xxManager**	**getInstance**(Context parentContext) This method initialises an application, obtaining a value of D2xx device manager.
static int	**getLibraryVersion**() This method returns the D2XX library version number.
int[][]	**getVIDPID**() This retrieves the current VID and PID combination from within the internal device list table.
boolean	**isFtDevice**(UsbDevice dev) This queries if a plugged-in USB device is a valid FT_Device
FT_Device	**openByDescription**(Context parentContext, java.lang.String description) This designates the device with the specified description.
FT_Device	**openByDescription**(Context parentContext, java.lang.String description, **D2xxManager.DriverParameters** params) This designates the device with the specified description and allows for configuration of driver parameters.
FT_Device	**openByIndex**(Context parentContext, int index) This designates the device at the specified index.
FT_Device	**openByIndex**(Context parentContext, int index, **D2xxManager.DriverParameters** params) This designates the device at the specified index and allows for configuration of driver parameters.
FT_Device	**openByLocation**(Context parentContext, int location) This designates the device at the specified location.
FT_Device	**openByLocation**(Context parentContext, int location, **D2xxManager.DriverParameters** params) This designates the device at the specified location, and allows for configuration of driver parameters.
FT_Device	**openBySerialNumber**(Context parentContext, java.lang.String serialNumber) This designates the device with the specified serial number.
FT_Device	**openBySerialNumber**(Context parentContext, java.lang.String serialNumber, **D2xxManager.DriverParameters** params) This designates the device with the specified serial number for use, and allows for configuration of driver parameters.
FT_Device	**openByUsbDevice**(Context parentContext, UsbDevice dev) This designates the device from the specified USB Device object..
FT_Device	**openByUsbDevice**(Context parentContext, UsbDevice dev, **D2xxManager.DriverParameters** params) This designates the device from the specified USB Device object, and allows for configuration of driver parameters.
boolean	**setVIDPID**(int vendorId, int productId) This allows a custom VID and PID combination within the internal device list table.

4.1 createDeviceInfoList

Definition:

public int **createDeviceInfoList**(Context parentContext)

Summary:

This method builds an internal device information list and returns the number of D2XX devices connected to the system. The list contains information about both unopened and opened devices. Device information may be retrieved via the getDeviceInfoList(int, com.ftdi.j2xx.D2xxManager.FtDeviceInfoListNode[]) or getDeviceInfoListDetail(int) methods.

Remarks:

An application can use this function to ascertain the number of devices attached to the system.The application allocates space for the device information list and retrieves the list using getDeviceInfoList(int, com.ftdi.j2xx.D2xxManager.FtDeviceInfoListNode[]) or getDeviceInfoListDetail(int) methods. . If the devices connected to the system change, the device info list will not be updated until "createDeviceInfoList"(Context) is called again.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

Returns:

The number of devices represented in the device information list. This is used to ensure that sufficient storage for the device list is returned by getDeviceInfoList(int, com.ftdi.j2xx.D2xxManager.FtDeviceInfoListNode[]).

4.2 getDeviceInfoList

Definition:

public int **getDeviceInfoList**(int numDevs, D2xxManager.FtDeviceInfoListNode[] deviceList)

Summary:

This method returns the device list created with a prior call to createDeviceInfoList(Context). The list contains all available information for all the available devices at the time that createDeviceInfoList(Context) was called.

NOTE: The flags element of each FtDeviceInfoListNodeobject in the list is a bit-mask of FT_FLAGS_OPENED and FT_FLAGS_HI_SPEED.

Remarks:

This function should only be called after calling createDeviceInfoList(Context). If the devices connected to the system change, the device info list will not be updated until createDeviceInfoList(Context) is

called again. Location ID information is not returned for devices that are open when createDeviceInfoList(Context) is called. Information is not available for devices which are open in other processes. The Flags parameter of FT_FLAGS_OPENED indicates that the device is open, with the other fields being unpopulated.

Parameters:

numDevs - The number of devices represented in the device information list.

deviceList - An array of FtDeviceInfoListNode. That contains information on all available devices after a successful call.

Returns:

The number of devices represented in the device information list as returned from the getDeviceInfoList(int, com.ftdi.j2xx.D2xxManager.FtDeviceInfoListNode[]) call.

4.3 getDeviceInfoListDetail:

Definition:

public D2xxManager.FtDeviceInfoListNode **getDeviceInfoListDetail**(int index)

Summary:

This method returns information for a single device from the internal device list created by a previous call to createDeviceInfoList(Context). The flags element of the FtDeviceInfoListNode object is a bit-mask of FT_FLAGS_OPENED and FT_FLAGS_HI_SPEED.

NOTE: This function is to be called after calling createDeviceInfoList(Context). The device info list is not updated where changes are made to the connected devices until createDeviceInfoList(Context) is called again. The index value is zero-based.

Parameters:

index - An index of the information pertaining to the devices in the list.

Returns:

A FtDeviceInfoListNode object containing the information available for the device at the specified index in the list. NULL for error.

4.4 getInstance

Definition:

public static D2xxManager **getInstance**(Context parentContext)

Summary:

This method initialises an application, obtaining a value of D2xx device manager.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

Returns:

An instance of theD2xx device manager.

4.5 getLibraryVersion

Definition:

public static int **getLibraryVersion**()

Summary:

This method returns the D2XX library version number.

Returns:

A 32-bit number representing the library version in binary coded decimal format.

4.6 setVIDPID

Definition:

public boolean setVIDPID(int vendorId, int productId)

Summary:

This allows a custom VID and PID combination to be added within the internal device list table. This loads the drivers for the specified VID and PID combination.

NOTE:

The default driver supports a limited set of VID and PID matched devices.

NOTE:
The driver supports a limited set of VID and PID matched devices.
VID : 0x0403 , FTDI
PID : 0x6015 , X Series Device
PID : 0x6014 , FT232H Device
PID : 0x6011 , FT4232H Device
PID : 0x6010 , FT2232 or FT2232H Device
PID : 0x6001 , 232AM, FT232B or FT232R
PID : 0x6006 , Direct Driver Recovery PID
PID : 0xFAC1 , USB Instruments PS40M10
PID : 0xFAC2 , USB Instruments DS1M12
PID : 0xFAC3 , USB Instruments DS100M10
PID : 0xFAC4 , USB Instruments DS60M10
PID : 0xFAC5 , USB Instruments EasySYNC LA100
PID : 0xFAC6 , USB2-F-7x01 CANPlus Adapter
PID : 0x6012 , ES001H
PID : 0x1025 , Macraigor - customer request
PID : 0x0001 , Keith Support Request 8/10/04
PID : 0x6017 , Additional VID/PID). To use this driver with other VID and PID combinations, the setVIDPID function is a pre-requisite.

openByIndex(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters), openByLocation(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters), openBySerialNumber(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters), openByDescription(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters), createDeviceInfoList(Context).

Parameters:

vendorId - The vendor ID that the driver aligns with

productId - The product ID that the driver aligns with

Returns:

If success , return true.

4.7 getVIDPID

Definition:

public int[][] **getVIDPID**()

Summary:

This retrieves the current VID and PID combination from within the internal device list table. The VID and PID can be matched using setVIDPID(int, int)

Returns:

2-element array containing the VID in the first element and the PID in the second element.

4.8 isFtDevice

Definition:

public boolean **isFtDevice**(UsbDevice dev)

Summary:

This queries if a plugged-in USB device is a valid FT_Device

Parameters:

dev - The UsbDevice get from ACTION_USB_DEVICE_ATTACHED broadcast.

Returns:

If the plugged in USB device is ascertained to be a valid FT device, the query returns a 'true' value

4.9 openByDescription with DriverParameters

Definition:

public FT_Device **openByDescription**(Context parentContext,java.lang.String description, D2xxManager.DriverParameters params)

Summary:

This designates the device with the specified description and allows for configuration of driver parameters.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

description - Description of the device.

DriverParameters - Parameters to configure max transfer size, buffer size, Rx timeout and number of buffers.

Buffer size: the default is 16k , Max is 16k , Min is 64.

Packet size : the default is 16384 , Max is 16384 , Min is 64.
Buffer Number : the default is 16 , Max is 16 , Min is 2.

Returns:

A FT_Device object containing the device object, 'NULL'if there is an error

4.10 openByDescription without DriverParameters

Definition:

public FT_Device **openByDescription**(Context parentContext,

java.lang.String description)

Summary:

This designates the device with the specified description.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

description - Description of the device.

Returns:

A FT_Device object containing the device object, NULL for error.

4.11 openByIndex with DriverParameters

Definition:

public FT_Device **openByIndex**(Context parentContext,int index,

D2xxManager.DriverParameters params)

Summary:

This designates the device at the specified index and allows for configuration of driver parameters.
Parameters:

parentContext - The calling activity must pass the application Context into this function.index - The index of the device, which is 0 based..

DriverParameters - Parameters to configure max transfer size, buffer size, Rx timeout and number of buffers.

Buffer size : the default is 16k , Max is 16k , Min is 64.

Packet size : the default is 16384 , Max is 16384 , Min is 64.

Buffer Number : the default is 16 , Max is 16 , Min is 2.

Returns:

A FT_Device object containing the device object, NULL for error

4.12 openByIndex without DriverParameters

Definition:

public FT_Device **openByIndex**(Context parentContext, int index)

Summary:

This designates the device at the specified index.
Parameters:

parentContext - The calling activity must pass the application Context into this function.

index - The index of the device , which is 0 based.

Returns:

A FT_Device object containing the device object, NULL for error

4.13 openByLocation with DriverParameters

Definition:

public FT_Device **openByLocation**(Context parentContext, int location,
D2xxManager.DriverParameters params)

Summary:

This designates the device at the specified location, and allows for configuration of driver parameters.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

location - The location of the device.

DriverParameters - Parameters to configure max transfer size, buffer size, Rx timeout and number of buffers.

Buffer size : the default is 16k , Max is 16k , Min is 64.

Packet size : the default is 16384 , Max is 16384 , Min is 64.

Buffer Number : the default is 16 , Max is 16 , Min is 2.

Returns:

A FT_Device object containing the device object, NULL for error

4.14 openByLocation without DriverParameters

Definition:

public FT_Device **openByLocation**(Context parentContext,int location)

Summary:

This designates the device at the specified location

Parameters:

parentContext - The calling activity must pass the application Context into this function.

location - The location of the device.

Returns:

A FT_Device object containing the device object, NULL for error

4.15 openBySerialNumber with DriverParameters

Definition:

public FT_Device **openBySerialNumber**(Context parentContext,

java.lang.StringserialNumber, D2xxManager.DriverParameters params)

Summary:

This designates the device with the specified serial number for use, and allows for configuration of driver parameters.

Parameters:

parentContext - Calls this function

serialNumber - The serial number of the device.

DriverParameters - Parameters to configure max transfer size, buffer size, Rx timeout and number of buffers.

Buffer size : the default is 16k , Max is 16k , Min is 64.

Packet size : the default is 16384 , Max is 16384 , Min is 64.

Buffer Number : the default is 16 , Max is 16 , Min is 2.

Returns:

A FT_Device object containing the device object, NULL for error

4.16 openBySerialNumber without DriverParameters

Definition:

public FT_Device **openBySerialNumber**(Context parentContext,

java.lang.StringserialNumber)

Summary:

This designates the device with the specified serial number.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

serialNumber - The serial number of the device.

Returns:

A FT_Device object containing the device object, NULL for error

4.17 openByUSBDevice with DriverParameters

Definition:

public FT_Device **openByUsbDevice**(Context parentContext, UsbDevice dev, D2xxManager.DriverParameters params)

Summary:

This designates the device from the specified USB Device object, and allows for configuration of driver parameters.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

dev - The USB Device object as an FT_Device

DriverParameters - Parameters to configure max transfer size, buffer size, Rx timeout and number of buffers.

Buffer size : the default is 16k , Max is 16k , Min is 64.

Packet size : the default is 16384 , Max is 16384 , Min is 64.

Buffer Number : the default is 16 , Max is 16 , Min is 2.

Returns:

A FT_Device object containing the device object, NULL for error

4.18 openByUSBDevice without DriverParameters

Definition:

public FT_Device **openByUsbDevice**(Context parentContext,

UsbDevicedev)

Summary:

This designates the device from the specified USB Device object.

Parameters:

parentContext - The calling activity must pass the application Context into this function.

dev - The USB Device object as an FT_Device

Returns:

A FT_Device object containing the device object, NULL for error

4.19 Sample

This is a sample show how to configure FT device to UART mode, please refer to sample project for more information.

```
public class sample extends Activity {
  public static D2xxManager ftD2xx= null;
  FT_Device ftDev = null;
  int devCount = 0;
  @override
  public void onCreate(Bundle savedInstanceState) {
            try {
                      // Get FT_Device and Open the port
                      ftD2xx = D2xxManager.getInstance(this);
                      devCount = ftdid2xx.createDeviceInfoList(this);
                      if (devCount> 0) {
                                ftDev = ftdid2xx.openByIndex(this, index);
                      }
                      // Configure the port to UART
                      If( ftDev.isOpen() == true ) {
                                // Reset FT Device
                                ftDev.setBitMode((byte)0 , D2xxManager.FT_BITMODE_RESET);
                                // Set Baud Rate
                                ftDev.setBaudRate(115200);
                                // Set Data Bit , Stop Bit , Parity Bit
                                ftDev.setDataCharacteristics(D2xxManager.FT_DATA_BITS_8,
                                D2xxManager.FT_STOP_BITS_1, D2xxManager.FT_PARITY_NONE);
                                // Set Flow Control
                                ftDev.setFlowControl(D2xxManager.FT_FLOW_NONE, (byte) 0x0b, (byte)
                                0x0d);
                      }
  } catch (D2xxManager.D2xxException ex) {
            ex.printStackTrace();
            ftDev.close();
  }
}
```

5 FieldsinD2xxManager Class

5.1 Data Bits<byte>

FT_DATA_BITS_7 = 7

FT_DATA_BITS_8 = 8

5.2 Stop Bits<byte>

FT_STOP_BITS_1 = 0
FT_STOP_BITS_2 = 2

5.3 Parity Bits< byte>

FT_PARITY_NONE	= 0
FT_PARITY_ODD	= 1
FT_PARITY_EVEN	= 2
FT_PARITY_MARK	= 3
FT_PARITY_SPACE	= 4

5.4 Flow Control Bits<short>

FT_FLOW_NONE	= 0x0000
FT_FLOW_RTS_CTS	= 0x0100
FT_FLOW_DTR_DSR	= 0x0200
FT_FLOW_XON_XOFF	= 0x0400

5.5 Purge Flags<byte>

FT_PURGE_RX	= 1
FT_PURGE_TX	= 2

5.6 Modem StatusBits<byte>

FT_CTS	= 0x10
FT_DSR	= 0x20
FT_RI	= 0x40
FT_DCD	= 0x80

5.7 Line Status Bits<byte>

FT_OE	= 0x02
FT_PE	= 0x04
FT_FE	= 0x08
FT_BI	= 0x10

5.8 Event Bits<byte>

Rx Char Event :

FT_EVENT_RXCHAR	= 0x01

Modem Status Event:

FT_EVENT_MODEM_STATUS	= 0x02

Line Status Event:

FT_EVENT_LINE_STATUS	= 0x04

Removed Event:

FT_EVENT_REMOVED	= 0x08

5.9 Device Information List Flags<byte>

Device Open Flag:

FT_FLAGS_OPENED	=1

Device Hi Speed Flag:

FT_FLAGS_HI_SPEED	= 2

5.10 Device Type< int>

bvdDevice = 0x0200 or 0x0400 and iSerialNumber != 0:

FT_DEVICE_232B	= 0

bvdDevice = 0x0200 and iSerialNumber != 0 :

FT_DEVICE_8U232AM	= 1

Unknown :

FT_DEVICE_UNKNOWN	= 3

bvdDevice = 0x0500 :

FT_DEVICE_2232	= 4

bvdDevice = 0x0600 :

FT_DEVICE_232R	= 5

bvdDevice = 0x0600 :

FT_DEVICE_245R	= 5

bvdDevice = 0x0700 :

FT_DEVICE_2232H	= 6

bvdDevice = 0x0800 :

FT_DEVICE_4232H	= 7

bvdDevice = 0x0900 :

FT_DEVICE_232H	= 8

bvdDevice = 0x1000 :

FT_DEVICE_X_SERIES	= 9

5.11 Bit Mode Bits<byte>

Reset:

FT_BITMODE_RESET	= 0x00

Asynchronous Bit Bang:

FT_BITMODE_ASYNC_BITBANG	= 0x01

MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only) :

FT_BITMODE_MPSSE	= 0x02

Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) :

FT_BITMODE_SYNC_BITBANG	= 0x04

MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only) :

FT_BITMODE_MCU_HOST	= 0x08

Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) :

FT_BITMODE_FAST_SERIAL	= 0x10

CBUS Bit Bang Mode (FT232R and FT232H devices only) :

FT_BITMODE_CBUS_BITBANG	= 0x20

Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only) :

FT_BITMODE_SYNC_FIFO	= 0x40

5.12 Break On Bits<int>

UART break on condition:

FTDI_BREAK_OFF = 0x0000,
UART break off condition:

FTDI_BREAK_ON = 0x4000

6 D2xxManager.FtDeviceListNode

Information about a connected FTDI device. The D2xxManager.getDeviceInfoListDetail(int) returns one of these structures; the D2xxManager.getDeviceInfoList(int, com.ftdi.j2xx.D2xxManager.FtDeviceInfoListNode[]) method returns an array of these structures.

6.1 Fields

Modifier and Type	Field and Description
short	**bcdDevice** Indicate USB specification release number(BCD).
int	**breakOnParam** Indicate breakOnParam variable to representation UART break status Default value is 0x0008; Device can set break on via **FT_Device.setBreakOn()** method.
String	**description** Description string for FT device, if available.
int	**flags** Indicates if device is already open (**D2xxManager.FT_FLAGS_OPENED**), or supports hi-speed (**D2xxManager.FT_FLAGS_HI_SPEED**).
int	**handle** Reserve
int	**id** Reserve
byte	**iSerialNumber** The iSerialNumber field on the USB Device Descriptor.
short	**lineStatus** Indicate lineStatus variable to representation UART line status Device can get line status via **FT_Device.getLineStatus()** method.
int	**location** The location number for FT device, if available.
short	**modemStatus** Indicate modemStatus variable to representation UART modem status Device can get modem status via **FT_Device.getModemStatus()** method.
String	**serialNumber** Serial number string for FT device, if available.

int	**type** Identifies this device in the FTDI family, such as **D2xxManager.FT_DEVICE_232R** or **D2xxManager.FT_DEVICE_X_SERIES**.

Field Detail

flags

public int flags

Indicates if device is already open (D2xxManager.FT_FLAGS_OPENED), or supports hi-speed (D2xxManager.FT_FLAGS_HI_SPEED).

bcdDevice

public short bcdDevice

Indicate USB specification release number(BCD).

type

public int type

Identifies this device in the FTDI family, such as D2xxManager.FT_DEVICE_232R or D2xxManager.FT_DEVICE_X_SERIES.

iSerialNumber

public byte iSerialNumber

iSerialNumber field on the USB Device Descriptor. Index of string descriptor for the serial number.

id

public int id

Reserve

location

public int location

location number for FT device, if available. Uniquely identifies the device and interface. This may change if another USB device is added to, or removed from, the computer. Device can be open via D2xxManager.openByLocation(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters) method.

serialNumber

public java.lang.String serialNumber

Serial number string for FT device, if available. Device can be open via D2xxManager.openBySerialNumber(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters) method.

description

public java.lang.String description

Description string for FT device, if available. Device can be open via D2xxManager.openByDescription(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters) method.

handle

public int handle

Reserve

breakOnParam

public int breakOnParam

Indicate breakOnParam variable to representation UART break status Default value is 0x0008; Device can set break on via FT_Device.setBreakOn() method. Device can set break off via FT_Device.setBreakOff() method.

modemStatus

public short modemStatus

Indicate modemStatus variable to representation UART modem status Device can get modem status via FT_Device.getModemStatus() method.

lineStatus

public short lineStatus

Indicate lineStatus variable to representation UART line status Device can get line status via FT_Device.getLineStatus() method.

6.2 Constructor

Constructor and Description
D2xxManager.FtDeviceInfoListNode()

Constructor Detail

D2xxManager.FtDeviceInfoListNode

public D2xxManager.FtDeviceInfoListNode()

7 D2xxManager.D2xxException

A class for handling D2xx exceptions and the printing of error messages.

7.1 Constructor

Constructor and Description
D2xxManager.D2xxException()   A constructor handling exception without any parameters
D2xxManager.D2xxException(java.lang.String ftStatusMsg)   A constructor handling exception with string parameters

Constructor Detail

D2xxManager.D2xxException

public D2xxManager.D2xxException()

A constructor handling exception without any parameters

D2xxManager.D2xxException

public D2xxManager.D2xxException(java.lang.String ftStatusMsg)

A constructor handling exception with string parameters

8 D2xxManager.DriverParameters

A management class for connected FTDI devices. Use getInstance(Context) to get a copy of D2xxManager; use createDeviceInfoList(Context) method to scan current connected FTDI devices, then open target device via a suitable open API.

openByDescription,

openByIndex,

openByLocation,)

openBySerialNumber,

openByUsbDevice,

8.1 Constructor

Constructor and Description
D2xxManager.DriverParameters() DriverParameters constructor Default Parameteres: Buffer Size : 16k Max Transfer Size : 16k Number Buffer : 16 Read Timeout : 5000 ms

8.2 Methods

Method Summary

Methods

Modifier and Type	Method and Description
int	getBufferNumber() This method will return Buffer number for Rx in user space application.
int	getMaxBufferSize() This method will return Rx buffer size of user space application.
int	getMaxTransferSize() This method will return Max Transfer size for Rx in the user space application.
Int	getReadTimeout() This method will return timeout values to be used for read operations.
Boolean	setBufferNumber(int number) This method will set the Buffer number for Rx in the user space application.
Boolean	setMaxBufferSize(int size) This method will set the Max Buffer size to process Rx data in the user space application.
Boolean	setMaxTransferSize(int size) This method will set the Max Transfer size to process Rx data in the user space application.
Boolean	setReadTimeout(int timeout) This method specifies the timeout values to be used for read operations.

8.2.1 getBufferNumber

Definition:

public int **getBufferNumber**()

Summary:

This returns the Buffer number for Rx in user space application.

Returns:

The current number of the Rx buffer.

8.2.2 setBufferNumber

Definition:

public boolean **setBufferNumber**(int number)

Summary:

This method sets the Buffer number for Rx in the user space application. The default is a minimum of 2 and a maximum of 16.

Parameters:

number - Specifies the value to Buffer Number

Returns:

If success , return true.

8.2.3 getMaxTransferSize

Definition:

public int **getMaxTransferSize**()

Summary:

This method will return Max Transfer size for Rx in the user space application.

Returns:

The current size of Rx Max Transfer

8.2.4 setMaxTransferSize

Definition:

public boolean **setMaxTransferSize**(int size)

Summary:

This method will set the Max Transfer size to process Rx data in the user space application. The default is 16384 , Max is 16384 , Min is 64.

Parameters:

size - Specifies the value of the Max Transfer size.

Returns:

If success , return true.

8.2.5 getMaxBufferSize

Definition:

public int **getMaxBufferSize**()

Summary:

This method will return Rx buffer size of user space application.

Returns:

The current size of Rx buffer.

8.2.6 setMaxBufferSize

Definition:

public boolean **setMaxBufferSize**(int size)

Summary:

This method will set the Max Buffer size to process Rx data in the user space application. The default is 16k , Max is 16K , Min is 64.

Parameters:

size - Specifies the value to Max BufferSize

Returns:

If success , return true.

8.2.7 getReadTimeout

Definition:

public int **getReadTimeout**()

Summary:

This method will return timeout values to be used for read operations.

Returns:

The current value (ms) of read timeout.

8.2.8 setReadTimeout

Definition:

public boolean **setReadTimeout**(int timeout)

Summary:

This method specifies the timeout values to be used for read operations. Default timeout values are 5000 mS which is interpreted as infinite; in this case read calls will block until all of the requested data has been received.

Parameters:

readTimeout - The value in mS to apply to read operations. Default is 5000 mS

Returns:

If success , return true.

9 FT_Device

The FT_Device class provides APIs for the host to communicate and operate FTDI devices. A typical use case would follow the below sequence:

1. Use getInstance to get a copy of D2xxManager
2. Use createDeviceInfoList method to scan current connected FTDI devices
3. Open target device to get FT_Device instance via a suitable open API.

Constructors

Constructor and Description
FT_Device(Context parentContext, UsbManager usbManager, UsbDevice u, UsbInterface i)

Method Summary

Methods

Modifier and Type	Method and Description
void	**close**() Closes a device opened with a previous call to **D2xxManager.openByIndex(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters)**, **D2xxManager.openBySerialNumber(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters)**, **D2xxManager.openByDescription(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters)** or **D2xxManager.openByLocation(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters)**.
boolean	**clrDtr**() Allows the DTR modem control line to be manually de-asserted.
boolean	**clrRts**() Allows the RTS modem control line to be manually de-asserted.
boolean	**eepromErase**() Erases the device EEPROM.
int	**eepromGetUserAreaSize**() Retrieves the amount of additional space available in the device EEPROM.
FT_EEPROM	**eepromRead**() Reads the entire device EEPROM and decodes its settings in to fields in a **FT_EEPROM** object. Remarks: **FT_EEPROM** : For FT_232A , FT_232B. **FT_EEPROM_2232H** : For FT_2232H. **FT_EEPROM_2232D** : For FT_2232. **FT_EEPROM_4232H** : For FT_4232H. **FT_EEPROM_232R** : For FT_232R. **FT_EEPROM_245R** : For FT_245R. **FT_EEPROM_232H** : For FT_232H. FT_EEPROM_X : For FT_X_Series.
byte[]	**eepromReadUserArea**(int length) Retrieves the contents of the device EEPROM user area.

int	**eepromReadWord**(short offset) Reads a WORD from the device EEPROM at the specified address.
short	**eepromWrite**(**FT_EEPROM**eeData) Encodes the settings from a **FT_EEPROM** object and writes them to the device EEPROM. Remarks: **FT_EEPROM** : For FT_232A , FT_232B. **FT_EEPROM_2232H** : For FT_2232H. **FT_EEPROM_2232D** : For FT_2232. **FT_EEPROM_4232H** : For FT_4232H. **FT_EEPROM_232R** : For FT_232R. **FT_EEPROM_245R** : For FT_245R. **FT_EEPROM_232H** : For FT_232H. FT_EEPROM_X : For FT_X_Series.
int	**eepromWriteUserArea**(byte[] data) Writes data to the device EEPROM user area.
boolean	**eepromWriteWord**(short address, short data) Writes a WORD to the device EEPROM at the specified address.
byte	**getBitMode**() Gets the instantaneous value of the data bus.
D2xxManager.FtDeviceInfoListNode	**getDeviceInfo**() Retrieves information on the device that is currently open.
long	**getEventStatus**() Retrieves the event status
byte	**getLatencyTimer**() Retrieves the current latency timer value from the device.
short	**getLineStatus**() Retrieves the current modem line status values for the device.
short	**getModemStatus**() Retrieves the current modem status values for the device.
int	**getQueueStatus**() Retrieves the number of bytes available to read from the Rx driver buffer.
boolean	**isOpen**() Returns the open status of the device.
boolean	**purge**(byte flags) Discards any data from the specified driver buffer and also removes data from the device.
int	**read**(byte[] data) Reads data from the device into the Java application buffer.
int	**read**(byte[] data, int length) Reads data from the device into the Java application buffer.
boolean	**readBufferFull**() Returns if the Rx buffer was full with data, if true, Rx would be pending until the data is read by user.
boolean	**resetDevice**() Sends a vendor command to the device to cause a reset and removes any data from the device buffers.
void	**restartInTask**() Restarts the driver's IN thread following a successful call to **stopInTask()** Remarks: This function restarts the driver's IN task (read) after it has been stopped by a call to **stopInTask()**.

boolean	**setBaudRate**(int baudRate) Sends a vendor command to the device to change the baud rate generator value.
boolean	**setBitMode**(byte mask, byte bitMode) Uses an alternative interface mode such as bit-bang, MPSSE and CPU target mode.
boolean	**setBreakOff**() Resets the BREAK condition on the device UART.
boolean	**setBreakOn**() Generates a BREAK condition on the device UART.
boolean	**setChars**(byte eventChar, byte eventCharEnable, byte errorChar, byte errorCharEnable) Specifies the event character and error replacement characters for the device.
boolean	**setDataCharacteristics**(byte dataBits, byte stopBits, byte parity) Dictates the data format that the device uses.
boolean	**setDtr**() Allows the DTR modem control line to be manually asserted.
boolean	**setEventNotification**(long Mask) Specifies events for the java driver to signal that they have occurred.
boolean	**setFlowControl**(short flowControl, byte xon, byte xoff) Specifies the flow control method that the device should use to prevent data loss.
boolean	**setLatencyTimer**(byte latency) Allows the latency timer value for the device to be specified.
boolean	**setRts**() Allows the RTS modem control line to be manually asserted.
void	**stopInTask**() Stops the driver's IN thread and prevents USB IN requests being issued to the device.
boolean	**stoppedInTask**() Return the running status of starts the driver's IN thread.
int	**write**(byte[] data) Writes data to the device from the Java application buffer.
int	**write**(byte[] data, int length) Writes data to the device from the Java application buffer.

9.1 close

Definition:

public void **close**()

Summary:

Closes a device opened with a previous call to **D2xxManager.openByIndex(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters)**, **D2xxManager.openBySerialNumber(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters)**, **D2xxManager.openByDescription(Context, java.lang.String, com.ftdi.j2xx.D2xxManager.DriverParameters)** or **D2xxManager.openByLocation(Context, int, com.ftdi.j2xx.D2xxManager.DriverParameters)**.

9.2 getDeviceInfo

Definition:

public D2xxManager.FtDeviceInfoListNode **getDeviceInfo**()

Summary:

Retrieves information on the device that is currently open.

Returns:

A FtDeviceInfoListNode object containing the information available for the device. Note that the flags and location fields are not used by this method.

9.3 getLineStatus

Definition:

public short **getLineStatus**()

Summary:

Retrieves the current modem line status values for the device.

NOTE this is only meaningful when the device is in UART mode.

Returns:

A short value containing the line status. The line status is a bit-mask of FT_OE, FT_PE, FT_FE and FT_BI. Negative value for error.

9.4 getModemStatus

Definition:

public short **getModemStatus**()

Summary:

Retrieves the current modem status values for the device.

NOTE: this is only meaningful when the device is in UART mode.

Returns:

A short value containing the modem status. The modem status is a bit-mask of FT_CTS, FT_DSR, FT_RI and FT_DCD. Negative value for error.

9.5 getQueueStatus

Definition:

public int **getQueueStatus**()

Summary:

Retrieves the number of bytes available to read from the driver Rx buffer.

Returns:

The number of bytes available in the driver Rx buffer. A call to read(byte[], int) requesting up to this number of bytes will return with the data immediately. Returns negative number for error.

9.6 isOpen

Definition:

public boolean **isOpen**()

Summary:

Returns the open status of the device

Returns:

Returns true if the device is open, false otherwise.

9.7 purge

Definition:

public boolean **purge**(byte flags)

Summary:

Discards any data form the specified driver buffer and also flushes data from the device.

Parameters:

flags - Specifies the queue to purge. flags is a bit-mask of FT_PURGE_RX and FT_PURGE_TX.

Returns:

Return true mean SUCCESS.

9.8 read with three parameters

Definition:

public int read(byte[] data, int length, long wait_ms)

Summary:

This method reads data from the device in to the Java application buffer. The device must be open to read data from it. This method allows user to specify a custom read timeout value in milliseconds unit.

Parameters:

data - A data buffer containing the bytes read from the device.

length - The number of bytes that the application is requesting to be read from the device.

wait_ms - A custom wait timeout value in ms.

Returns:

The number of bytes successfully read from the device.

9.9 read with two parameters

Definition:

public int **read**(byte[] data, int length)

Summary:

This method reads data from the device in to the Java application buffer. The device must be open to read data from it.

Parameters:

data - A data buffer containing the bytes read from the device.

length - The number of bytes that the application is requesting to be read from the device.

Returns:

The number of bytes successfully read from the device.

9.10 Read with one parameter

Definition:

public int **read**(byte[] data)

Summary:

Reads data from the device in to the Java application buffer. The device must be open to read data from it. Will attempt to read data.length bytes from the device.

Parameters:

data - A data buffer containing the bytes read from the device.

Returns:

The number of bytes successfully read from the device.

9.11 readBufferFull

Definition:

public boolean readBufferFull()

Summary:

This method return if the Rx buffer was full with data. If true, Rx would be pending until the data is read by user.

Returns:

True if Rx buffer is full.

9.12 write with two parameters

Definition:

public int **write**(byte[] data, int length)

Summary:

Writes data to the device from the Java application buffer. The device must be open to write data to it. This method will wait until USB request sent, then report how many bytes were written.

Parameters:

data - A data buffer containing the bytes to write to the device.

length - The number of bytes that the application is requesting to write to the device.

Returns:
The number of bytes successfully written to the device.

9.13 write with three parameters

Definition:

public int **write**(byte[] data, int length, boolean wait)

Summary:

Writes data to the device from the Java application buffer. The device must be open to write data to it. This method allows user to specify if one would like to wait for request sent to complete.

Parameters:

data - A data buffer containing the bytes to write to the device.

length - The number of bytes that the application is requesting to write to the device.

Returns:
The number of bytes successfully written to the device.

9.14 write with one parameter

Definition:

public int **write**(byte[] data)

Summary:

This method writes data to the device from the Java application buffer. The device must be open to write data to it. This method will wait until USB request sent is complete, and then report how many bytes were written.

Parameters:

data - A data buffer containing the bytes to write to the device.

Returns:

The number of bytes successfully written to the device.

9.15 resetDevice

Definition:

public boolean **resetDevice**()

Summary:

This method sends a vendor command to the device to cause a reset and flush any data from the device buffers.

Returns:

Return true mean success.

9.16 restartInTask

Definition:

public void **restartInTask**()

Summary:

Restarts the driver's IN thread following a successful call to stopInTask()

Remarks:

This function is used to restart the driver's IN task (read) after it has been stopped by a call to stopInTask().

9.17 stopInTask

Definition:

public void **stopInTask**()

Summary:

This method stops the driver's IN thread and prevents USB IN requests being issued to the device. No data will be received from the device if the IN thread is stopped.

Remarks:

Used to put the driver's IN task (read) into a wait state. It can be used in situations where data is being received continuously, so that the device can be purged without more data being received. It is used together with restartInTask() which sets the IN task running again.

9.18 stoppedInTask

Definition:

public boolean **stoppedInTask**()

Summary:

This method return the running status of the driver's IN thread.

Remarks:

This function is used to query the driver's IN task status.

Returns:

Return true if the driver’s IN task is paused , false indicates that driver’s IN task is not running.

9.19 setBaudrate

Definition:

public boolean **setBaudRate**(intbaudRate)

Summary:

This method sends a vendor command to the device to change the baud rate generator value. Note that the baud rate is only meaningful when the device is in UART or bit-bang mode.

Parameters:

baudRate - The baud rate value to set for the device. This must be a value >184 baud. The maximum baud rate for full speed devices is 3Mbaud, for hi-speed devices it is 12Mbaud.

Returns:

Return true mean success.

9.20 setBitMode

Definition:

public boolean **setBitMode**(byte mask, byte bitMode)

Summary:

Allows the device to use alternative interface modes such as bit-bang, MPSSE and CPU target mode. Note that not all modes are available on all devices; please consult the device data sheet for more information.

Remark:

For a description of available bit modes for the FT232R, see the application note "Bit Bang Modes for the FT232R and FT245R".

For a description of available bit modes for the FT2232, see the application note "Bit Mode Functions for the FT2232".

For a description of Bit Bang Mode for the FT232B and FT245B, see the application note "FT232B/FT245B Bit Bang Mode".

Application notes are available for download from the FTDI website.

Note that to use CBUS Bit Bang for the FT232R, the CBUS must be configured for CBUS Bit Bang in the EEPROM.

Note that to use Single Channel Synchronous 245 FIFO mode for the FT2232H, channel A must be configured for FT245 FIFO mode in the EEPROM.

Parameters:

mask - Bit-mask that specifies which pins are input (0) and which are output (1). Required for bit-bang modes.

In the case of CBUS bit-bang, the upper nibble of this value controls which pins are inputs and outputs, while the lower nibble controls which of the outputs are high and low.

bitMode - The desired device mode. This can be one of the following: FT_BITMODE_RESET, FT_BITMODE_ASYNC_BITBANG, FT_BITMODE_MPSSE, FT_BITMODE_SYNC_BITBANG, FT_BITMODE_MCU_HOST, FT_BITMODE_FAST_SERIAL, FT_BITMODE_CBUS_BITBANG or FT_BITMODE_SYNC_FIFO.

Returns:

Return true mean SUCCESS.

9.21 getBitMode

Definition:

public byte **getBitMode**()

Summary:

Gets the instantaneous value of the data bus.

Remark:

For a description of available bit modes for the FT232R, see the application note "Bit Bang Modes for the FT232R and FT245R".

For a description of available bit modes for the FT2232, see the application note "Bit Mode Functions for the FT2232".

For a description of bit bang modes for the FT232B and FT245B, see the application note "FT232B/FT245B Bit Bang Mode".

For a description of bit modes supported by the FT4232H and FT2232H devices, please see the IC data sheets.

These application notes are available for download from the FTDI website.

Returns:

The value read from the device pins. Negative value for error

9.22 setBreakOff

Definition:

public boolean **setBreakOff**()

Summary:

This method resets the BREAK condition on the device UART. Note that this method is only meaningful when the device is in UART mode.

Returns:

Return true mean success.

9.23 setBreakOn

Definition:

public boolean **setBreakOn**()

Summary:

This method generates a BREAK condition on the device UART. Note that this method is only meaningful when the device is in UART mode.

Returns:

Return true mean success.

9.24 setChar

Definition:

public boolean **setChars**(byte eventChar, byte eventCharEnable, byte errorChar,byteerrorCharEnable)

Summary:

Specifies the event character and error replacement characters for the device to use. When the device detects an event character being received, this will trigger an IN to the USB Host regardless of the number of bytes in the device's buffer or the latency timer value. When the device detects an error (FT_OE, FT_PE, FT_FE orFT_BI), the error character will be inserted in to the data stream to the USB host.

Parameters:

eventChar - The character for which the device to trigger an IN.

eventCharEnable - Enable or disable the use of the event character.

errorChar - The character that will be inserted in the data stream on the detection of an

error.errorCharEnable - Enable or disable the use of the error replacement character.

Returns:

Return 0 mean success.

9.25 setDataCharacteristics

Definition:

public boolean **setDataCharacteristics**(byte dataBits, byte stopBits, byte parity)

Summary:

This method dictates the data format that the device will use. Communication errors will occur if these parameters do not match those used by the external system Note that these data characteristics are only meaningful when the device is in UART mode.

Parameters:

dataBits - Valid data bit values are FT_DATA_BITS_7 or FT_DATA_BITS_8.

stopBits - Valid stop bit values are FT_STOP_BITS_1 or FT_STOP_BITS_2.

parity - Valid parity values are FT_PARITY_NONE, FT_PARITY_ODD, FT_PARITY_EVEN, FT_PARITY_MARK or FT_PARITY_SPACE.

Returns:

Return true mean success

9.26 setEventNotifcation

Definition:

public boolean **setEventNotification**(long Mask)

Summary:

This method specifies events for the java driver to signal that they have occurred. Once the event mask has been set.

Remarks:

An application uses this function to setup conditions which allow a thread to block until one of the conditions is met. Typically, an application will create an event, call this function, and then block on the event. When the conditions are met, the event is set, and the application thread unblocked. mask is a bit-map that describes the events the application is interested in. If one of the event conditions is met, the event is set. If FT_EVENT_RXCHAR is set in mask, the event will be set when a character has been received by the device. If FT_EVENT_MODEM_STATUS is set in mask, the event will be set when a change in the modem signals has been detected by the device. If FT_EVENT_LINE_STATUS is set in mask, the event will be set when a change in the line status has been detected by the device.

Parameters:

mask - Specifies the events to wait on. This is a bit-mask of FT_EVENT_RXCHAR, FT_EVENT_MODEM_STATUS and FT_EVENT_LINE_STATUS

Returns:

Return the event number.

9.27 getEventStatus

Definition:

public long **getEventStatus**()

Summary:

This method retrieves the event status

Returns:

The event status, negative for error

9.28 setFlowControl

Definition:

public boolean **setFlowControl**(short flowControl,bytexon, byte xoff)

Summary:

Specifies the flow control method that the device should use to prevent data loss.

Parameters:

flowControl - Valid flow control values are FT_FLOW_NONE, FT_FLOW_RTS_CTS, FT_FLOW_DTR_DSR or FT_FLOW_XON_XOFF.

xon - Specifies the character to use for XOn if FT_FLOW_XON_XOFF is enabled.

xoff - Specifies the character to use for XOff if FT_FLOW_XON_XOFF is enabled.

Returns:

Return true mean success

9.29 setLatencyTimer

Definition:

public boolean **setLatencyTimer**(byte latency)

Summary:

This method allows the latency timer value for the device to be specified. The latency timer is the mechanism that returns short packets to the USB host. The default value is 16ms.

Parameters:

In the FT8U232AM and FT8U245AM devices, the receive buffer timeout that is used to remove remaining data from the receive buffer is fixed at 16 ms. In other FTDI devices, this timeout is programmable and can be set at 1 ms intervals between 2ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response times from short data packets.

Remarks:

latency - The new value to use for the latency timer. The valid range for this is 2ms - 255ms.

Returns:

Return true mean success.

9.30 getLatencyTimer

Definition:

public byte **getLatencyTimer**()

Summary:

This method retrieves the current latency timer value from the device. The latency timer is the mechanism that returns short packets to the USB host. The default value is 16ms.

Remark:

In the FT8U232AM and FT8U245AM devices, the receive buffer timeout that is used to flush remaining data from the receive buffer was fixed at 16 ms. In all other FTDI devices, this timeout is programmable and can be set at 1 ms intervals between 2ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response times from short data packets.

Returns:

Return true mean success.

9.31 setDtr

Definition:

public boolean **setDtr**()

Summary:

Allows the DTR modem control line to be manually asserted. Note that this method is only meaningful when the device is in UART mode.

Returns:

Return true mean success.

9.32 clrDtr

Definition:

public boolean **clrDtr**()

Summary:

Allows the DTR modem control line to be manually de-asserted. Note that this method is only meaningful when the device is in UART mode.

Returns:

Return true mean success.

9.33 setRts

Definition:

public boolean **setRts**()

Summary:

Allows the RTS modem control line to be manually asserted. Note that this method is only meaningful when the device is in UART mode.

Returns:

Return true mean success.

9.34 clrRts

Definition:

public boolean **clrRts**()

Summary:

Allows the RTS modem control line to be manually de-asserted. Note that this method is only meaningful when the device is in UART mode.

Returns:

Return true mean success.

9.35 eepromErase

Definition:

public boolean **eepromErase**()

Summary:

Erases the device EEPROM. After erasing, all values read will be 0xFFFF.

NOTE: The FT232R, FT245R and X-Series devices cannot have their EEPROMs erased as the EEPROM is internal to the device.

Returns:

Returns true on success, false otherwise.

9.36 eepromRead

Definition:

public FT_EEPROM eepromRead()

Summary:

This method reads the entire device EEPROM and decodes its settings in to fields in a FT_EEPROM object.

Remarks:

FT_EEPROM : For FT_232A , FT_232B.

FT_EEPROM_2232H : For FT_2232H.

FT_EEPROM_2232D : For FT_2232.

FT_EEPROM_4232H : For FT_4232H.

FT_EEPROM_232R : For FT_232R.

FT_EEPROM_245R : For FT_245R.

FT_EEPROM_232H : For FT_232H.

FT_EEPROM_X : For FT_X_Series.

Returns:

A FT_EEPROM object containing the parsed EEPROM settings for the device, NULL for error. FT_EEPROM can be cast to the actual device type.

9.37 eepromWrite

Definition:

public short eepromWrite(FT_EEPROM eeData)

Summary:

This method encodes the settings from a FT_EEPROM object and writes them to the device EEPROM.

Remarks:

FT_EEPROM : For FT_232A , FT_232B.

FT_EEPROM_2232H : For FT_2232H.

FT_EEPROM_2232D : For FT_2232.

FT_EEPROM_4232H : For FT_4232H.

FT_EEPROM_232R : For FT_232R.

FT_EEPROM_245R : For FT_245R.

FT_EEPROM_232H : For FT_232H.

FT_EEPROM_X : For FT_X_Series.

Parameters:

eeData - A FT_EEPROM object containing the EEPROM settings to be written to the device. FT_EEPROM can be cast to the actual device type

Returns:

Return 0 mean SUCCESS

9.38 eepromReadWord

Definition:
public int **eepromReadWord**(short offset)

Summary:

Reads a WORD from the device EEPROM at the specified address.

Parameters:

address - The EEPROM address to read from.

Returns:

The EEPROM data WORD read from the specified address. Negative value for error.

9.39 eepromWriteWord

Definition:

public boolean **eepromWriteWord**(short address, short data)

Summary:

Writes a WORD to the device EEPROM at the specified address.

Parameters:

address - The EEPROM address to write the new data to.data - The data WORD to write to the EEPROM at the address specified.

Returns:

Return true mean success

9.40 eepromGetUserAreaSize

Definition:

public int **eepromGetUserAreaSize**()

Summary:

Retrieves the amount of additional space available in the device EEPROM. This space (the user area) can be used to store application specific data.

Returns:

The number of unused EEPROM bytes available to the user. Negative value for error

9.41 eepromReadUserArea

Definition:

public byte[] **eepromReadUserArea**(int length)

Summary:

Retrieves the contents of the device EEPROM user area. The number of bytes returned matches the user area size returned from eepromGetUserAreaSize()

Parameters:

length - The length of word is read

Returns:

An array of bytes containing the user area data from the device EEPROM.NULL for error.

9.42 eepromWriteUserArea

Definition:

public int **eepromWriteUserArea**(byte[] data)

Summary:

Writes data to the device EEPROM user area. Once written, the data can be retrieved with a call to eepromReadUserArea(int).

Parameters:

data - The data to be written to the device EEPROM user area. The data is truncated if the size of data is greater than the space available in the EEPROM user area.

Returns:

if write success will return length of data , else 0

10 EEPROM Information

10.1 Class FT_EEPROM

EEPROM data structure on the 232A, 232B

10.1.1 Constructor

Constructor Summary

Constructors

Constructor and Description
FT_EEPROM()

10.1.2 Fields

Field Summary

Fields

Modifier and Type	Field and Description
Short	**DeviceType** Hardware Option - Invert RTS Signal
java.lang.String	**Manufacturer** String Descriptor - Manufacturer String
Short	**MaxPower** Configure Descriptor - Max USB Power Value between 0 and 500
java.lang.String	**Product** String Descriptor - Product String
Short	**ProductId** Device Descriptor - Product ID
boolean	**PullDownEnable** Hardware Option - Pull Down In Suspend Enabled
boolean	**RemoteWakeup** String Descriptor - Remote Wakeup Enabled
boolean	**SelfPowered** Configure Descriptor - Self Powered Mode
java.lang.String	**SerialNumber** String Descriptor - Serial Number String
boolean	**SerNumEnable** Device Descriptor - Serial Number Enabled
Short	**VendorId** Device Descriptor - Vendor ID

Field Detail

DeviceType

public short DeviceType

Hardware Option - Invert RTS Signal

Manufacturer

publicjava.lang.String Manufacturer

String Descriptor - Manufacturer String

Product

publicjava.lang.String Product

String Descriptor - Product String

SerialNumber

publicjava.lang.StringSerialNumber

String Descriptor - Serial Number String

VendorId

public short VendorId

Device Descriptor - Vendor ID

ProductId

public short ProductId

Device Descriptor - Product ID

SerNumEnable

public boolean SerNumEnable

Device Descriptor - Serial Number Enabled

MaxPower

public short MaxPower

Config Descriptor - Max USB Power Value between 0 and 500

SelfPowered

public boolean SelfPowered

Config Descriptor - Self Powered Mode

RemoteWakeup

public boolean RemoteWakeup

String Descriptor - Remote Wakeup Enabled

PullDownEnable

public boolean PullDownEnable

Hardware Option - Pull Down In Suspend Enabled

10.2 Class FT_EEPROM_232R

```
public class FT_EEPROM_232R
extends FT_EEPROM
```

EEPROM data structure on the 232R

10.2.1 Constructor

Constructor Summary

Constructors

Constructor and Description
FT_EEPROM_232R()

10.2.2 Fields

Field Summary

Fields

Modifier and Type	Field and Description
byte	**CBus0** Hardware Option - CBus0 Mux Control
byte	**CBus1** Hardware Option - CBus1 Mux Control
byte	**CBus2** Hardware Option - CBus2 Mux Control
byte	**CBus3** Hardware Option - CBus3 Mux Control
byte	**CBus4** Hardware Option - CBus4 Mux Control
boolean	**ExternalOscillator** Hardware Option - External Oscillator Caution: Setting this bit without an external oscillator fitted to your design will render the device unusable.
boolean	**HighIO** Drive Option - High Current IO
boolean	**InvertCTS** Hardware Option - Invert CTS signal
boolean	**InvertDCD** Hardware Option - Invert DCD signal
boolean	**InvertDSR** Hardware Option - Invert DSR signal
boolean	**InvertDTR** Hardware Option - Invert DTR signal

boolean	**InvertRI** Hardware Option - Invert RI signal
boolean	**InvertRTS** Hardware Option - Invert RTS signal
boolean	**InvertRXD** Hardware Option - Invert RXD signal
boolean	**InvertTXD** Hardware Option - Invert TXD signal
boolean	**LoadVCP** Driver Option - Load Virtual Com Port

Field Detail

HighIO

public boolean HighIO

Drive Option - High Current IO

ExternalOscillator

public boolean ExternalOscillator

Hardware Option - External Oscillator Caution: Setting this bit without an external oscillator fitted to your design renders the device unusable.

InvertTXD

public boolean InvertTXD

Hardware Option - Invert TXD signal

InvertRXD

public boolean InvertRXD

Hardware Option - Inverted RXD signal

InvertRTS

public boolean InvertRTS

Hardware Option - Invert RTS signal

InvertCTS

public boolean InvertCTS

Hardware Option - Invert CTS signal

InvertDTR

public boolean InvertDTR

Hardware Option - Invert DTR signal

InvertDSR

public boolean InvertDSR

Hardware Option - Invert DSR signal

InvertDCD

public boolean InvertDCD

Hardware Option - Invert DCD signal

InvertRI

public boolean InvertRI

Hardware Option - Invert RI signal

CBus0

public byte CBus0

Hardware Option - CBus0 Mux Control

CBus1

public byte CBus1

Hardware Option - CBus1 Mux Control

CBus2

public byte CBus2

Hardware Option - CBus2 Mux Control

CBus3

public byte CBus3

Hardware Option - CBus3 Mux Control

CBus4

public byte CBus4

Hardware Option - CBus4 Mux Control

LoadVCP

public boolean LoadVCP

Driver Option - Load Virtual Com Port

10.2.3Nested Class

```
public static final class FT_EEPROM_232R.CBUS

extends java.lang.Object
```

CBus Option on the FT232R

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	**FT_EEPROM_232R.CBUS** CBus Option on the FT232R

10.2.4CBUS Fields

FT 232R CBus Option	CBus Constant Variable(int)	Constant Value
TXDEN #	TXDEN	0
PWRON #	PWRON	1
RXLED #	RXLED	2
TXLED #	TXLED	3
TXRXLED #	TXRXLED	4
SLEEP #	SLEEP	5
48M Clock Hz #	CLK48MHz	6
24M Clock Hz #	CLK24MHz	7
12M Clock Hz #	CLK12MHz	8
6M Clock Hz #	CLK6MHz	9
IO_MODE #	IO_MODE	10
BIT_BANG_WR #	BIT_BANG_WR	11
BIT_BANG_RD #	BIT_BANG_RD	12

10.3 Class FT_EEPROM_245R

```
public class FT_EEPROM_245R
extends FT_EEPROM
```

EEPROM data structure on the 245R

10.3.1 Constructor

Constructor Summary

Constructors

Constructor and Description
FT_EEPROM_245R()

10.3.2 Fields

Field Summary

Fields

Modifier and Type	Field and Description
byte	**CBus0** Hardware Option - CBus0 Mux Control
byte	**CBus1** Hardware Option - CBus1 Mux Control
byte	**CBus2** Hardware Option - CBus2 Mux Control
byte	**CBus3** Hardware Option - CBus3 Mux Control
byte	**CBus4** Hardware Option - CBus4 Mux Control
boolean	**ExternalOscillator** Hardware Option - External Oscillator Caution: Setting this bit without an external oscillator fitted to your design renders the device unusable.
boolean	**HighIO** Drive Option - High Current IO
boolean	**InvertCTS** Hardware Option - Invert CTS signal
boolean	**InvertDCD** Hardware Option - Invert DCD signal
boolean	**InvertDSR** Hardware Option - Invert DSR signal
boolean	**InvertDTR** Hardware Option - Invert DTR signal
boolean	**InvertRI** Hardware Option - Invert RI signal

boolean	**InvertRTS** Hardware Option - Invert RTS signal
boolean	**InvertRXD** Hardware Option - Invert RXD signal
boolean	**InvertTXD** Hardware Option - Invert TXD signal
boolean	**LoadVCP** Driver Option - Load Virtual Com Port

Field Detail

HighIO

public boolean HighIO

Drive Option - High Current IO

ExternalOscillator

public boolean ExternalOscillator

Hardware Option - External Oscillator Caution: Setting this bit without an external oscillator fitted to your design renders the device unusable.

InvertTXD

public boolean InvertTXD

Hardware Option - Invert TXD signal

InvertRXD

public boolean InvertRXD

Hardware Option - Invert RXD signal

InvertRTS

public boolean InvertRTS

Hardware Option - Invert RTS signal

InvertCTS

public boolean InvertCTS

Hardware Option - Invert CTS signal

InvertDTR

public boolean InvertDTR

Hardware Option - Invert DTR signal

InvertDSR

public boolean InvertDSR

Hardware Option - Invert DSR signal

InvertDCD

public boolean InvertDCD

Hardware Option - Invert DCD signal

InvertRI

public boolean InvertRI

Hardware Option - Invert RI signal

CBus0

public byte CBus0

Hardware Option - CBus0 Mux Control

CBus1

public byte CBus1

Hardware Option - CBus1 Mux Control

CBus2

public byte CBus2

Hardware Option - CBus2 Mux Control

CBus3

public byte CBus3

Hardware Option - CBus3 Mux Control

CBus4

public byte CBus4

Hardware Option - CBus4 Mux Control

LoadVCP

public boolean LoadVCP

Driver Option - Load Virtual Com Port

10.3.3 Nested Class

```
public static final class FT_EEPROM_245R.CBUS
extends java.lang.Object
```

CBus Option on the FT245H

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	**FT_EEPROM_245R.CBUS** CBus Option on the FT245R

10.3.4 CBUS Fields

FT 245R CBUS Option	CBUS Constant Variable (int)	Constant Value
TXDEN #	TXDEN	0
PWRON #	PWRON	1
RXLED #	RXLED	2
TXLED #	TXLED	3
TXRXLED #	TXRXLED	4
SLEEP #	SLEEP	5
48M Clock Hz #	CLK48MHz	6
24M Clock Hz #	CLK24MHz	7
12M Clock Hz #	CLK12MHz	8
6M Clock Hz #	CLK6MHz	9
IO_MODE #	IO_MODE	10
BIT_BANG_WR #	BIT_BANG_WR	11
BIT_BANG_RD #	BIT_BANG_RD	12

10.4 Class FT_EEPROM_2232D

public class FT_EEPROM_2232D

extends FT_EEPROM

EEPROM data structure on the 2232D

10.4.1 Constructor

Constructors

Constructor and Description
FT_EEPROM_2232D()

10.4.2Fields

Field Summary

Fields

Modifier and Type	Field and Description
boolean	**A_FastSerial** Hardware Option - Interface A Fast Serial
boolean	**A_FIFO** Hardware Option - Interface A 245 FIFO
boolean	**A_FIFOTarget** Hardware Option - Interface A 245 FIFO CPU Target
boolean	**A_HighIO** Drive Option - Interface A High Current IO
boolean	**A_LoadD2XX** Driver Option - Interface A Load D2XX
boolean	**A_LoadVCP** Driver Option - Interface A Load Virtual Com Port
boolean	**A_UART** Drive Option - Interface A UART
boolean	**B_FastSerial** Hardware Option - Interface B Fast Serial
boolean	**B_FIFO** Hardware Option - Interface B 245 FIFO
boolean	**B_FIFOTarget** Hardware Option - Interface B 245 FIFO CPU Target
boolean	**B_HighIO** Drive Option - Interface B High Current IO
boolean	**B_LoadD2XX** Driver Option - Interface B Load D2XX
boolean	**B_LoadVCP** Driver Option - Interface B Load Virtual Com Port
boolean	**B_UART** Drive Option - Interface B UART

Field Detail

A_UART

public boolean A_UART

Drive Option - Interface A UART

B_UART

public boolean B_UART

Drive Option - Interface B UART

A_HighIO

public boolean A_HighIO

Drive Option - Interface A High Current IO

B_HighIO

public boolean B_HighIO

Drive Option - Interface B High Current IO

A_FIFO

public boolean A_FIFO

Hardware Option - Interface A 245 FIFO

B_FIFO

public boolean B_FIFO

Hardware Option - Interface B 245 FIFO

A_FIFOTarget

public boolean A_FIFOTarget

Hardware Option - Interface A 245 FIFO CPU Target

B_FIFOTarget

public boolean B_FIFOTarget

Hardware Option - Interface B 245 FIFO CPU Target

A_FastSerial

public boolean A_FastSerial

Hardware Option - Interface A Fast Serial

B_FastSerial

public boolean B_FastSerial

Hardware Option - Interface B Fast Serial

A_LoadVCP

public boolean A_LoadVCP

Driver Option - Interface A Load virtual Com Port

B_LoadVCP

public boolean B_LoadVCP

Driver Option - Interface B Load virtual Com Port

A_LoadD2XX

public boolean A_LoadD2XX

Driver Option - Interface A Load D2XX

B_LoadD2XX

public boolean B_LoadD2XX

Driver Option - Interface B Load D2XX

10.5 Class FT_EEPROM_2232H

public class **FT_EEPROM_2232H**
extends FT_EEPROM

EEPROM data structure on the 2232H

10.5.1 Constructor

Constructors

Constructor and Description
FT_EEPROM_2232H()

10.5.2Fields

Field Summary

Fields

Modifier and Type	Field and Description
boolean	**A_FastSerial** Hardware Option - Interface A Fast Serial
boolean	**A_FIFO** Hardware Option - Interface A 245 FIFO
boolean	**A_FIFOTarget** Hardware Option - Interface A 245 FIFO CPU Target
boolean	**A_LoadD2XX** Driver Option - Interface A Load D2XX Port
boolean	**A_LoadVCP** Driver Option - Interface A Load Vitrual Com Port
boolean	**A_UART** Hardware Option - Interface A 245 UART
byte	**AH_DriveCurrent** Drive Option - AH pins have drive current.
boolean	**AH_SchmittInput** Drive Option - AH pins have schmitt input.
boolean	**AH_SlowSlew** Drive Option - AH pins have slow slew.
byte	**AL_DriveCurrent**

	Drive Option - AL pins have drive current.
boolean	**AL_SchmittInput** Drive Option - AL pins have schmitt input.
boolean	**AL_SlowSlew** Drive Option - AL pins have slow slew.
boolean	**B_FastSerial** Hardware Option - Interface B Fast Serial
boolean	**B_FIFO** Hardware Option - Interface B 245 FIFO
boolean	**B_FIFOTarget** Hardware Option - Interface B 245 FIFO CPU Target
boolean	**B_LoadD2XX** Driver Option - Interface B Load D2XX Port
boolean	**B_LoadVCP** Driver Option - Interface B Load Vitrual Com Port
boolean	**B_UART** Hardware Option - Interface B 245 UART
byte	**BH_DriveCurrent** Drive Option - BH pins have drive current.
boolean	**BH_SchmittInput** Drive Option - BH pins have schmitt input.
boolean	**BH_SlowSlew** Drive Option - BH pins have slow slew.
byte	**BL_DriveCurrent** Drive Option - BL pins have drive current.
boolean	**BL_SchmittInput** Drive Option - BL pins have schmitt input.
boolean	**BL_SlowSlew** Drive Option - BL pins have slow slew.
boolean	**PowerSaveEnable** Hardware Option - Power Save Enable if using BCBus7 to save power for self-powered designs.
int	**TPRDRV**

Field Detail

AL_SlowSlew

public boolean AL_SlowSlew

Drive Option - AL pins have slow slew.

AL_SchmittInput

public boolean AL_SchmittInput

Drive Option - AL pins have schmitt input.

AL_DriveCurrent

public byte AL_DriveCurrent

Drive Option - AL pins have driver current.

AH_SlowSlew

public boolean AH_SlowSlew

Drive Option - AH pins have slow slew.

AH_SchmittInput

public boolean AH_SchmittInput

Drive Option - AH pins have schmitt input.

AH_DriveCurrent

public byte AH_DriveCurrent

Drive Option - AH pins have driver current.

BL_SlowSlew

public boolean BL_SlowSlew

Drive Option - BL pins have slow slew.

BL_SchmittInput

public boolean BL_SchmittInput

Drive Option - BL pins have schmitt input.

BL_DriveCurrent

public byte BL_DriveCurrent

Drive Option - BL pins have driver current.

BH_SlowSlew

public boolean BH_SlowSlew

Drive Option - BH pins have slow slew.

BH_SchmittInput

public boolean BH_SchmittInput

Drive Option - BH pins have schmitt input.

BH_DriveCurrent

public byte BH_DriveCurrent

Drive Option - BH pins have driver current.

A_UART

public boolean A_UART

Hardware Option - Interface A 245 UART

B_UART

public boolean B_UART

Hardware Option - Interface B 245 UART

A_FIFO

public boolean A_FIFO

Hardware Option - Interface A 245 FIFO

B_FIFO

public boolean B_FIFO

Hardware Option - Interface B 245 FIFO

A_FIFOTarget

public boolean A_FIFOTarget

Hardware Option - Interface A 245 FIFO CPU Target

B_FIFOTarget

public boolean B_FIFOTarget

Hardware Option - Interface B 245 FIFO CPU Target

A_FastSerial

public boolean A_FastSerial

Hardware Option - Interface A Fast Serial

B_FastSerial

public boolean B_FastSerial

Hardware Option - Interface B Fast Serial

PowerSaveEnable

public boolean PowerSaveEnable

Hardware Option - Power Save Enable if using BCBus7 to save power for self-powered designs.

A_LoadVCP

public boolean A_LoadVCP

Driver Option - Interface A Load virtual Com Port

B_LoadVCP

public boolean B_LoadVCP

Driver Option - Interface B Load virtual Com Port

A_LoadD2XX

public boolean A_LoadD2XX

Driver Option - Interface A Load D2XX Port

B_LoadD2XX

public boolean B_LoadD2XX

Driver Option - Interface B Load D2XX Port

TPRDRV

public int TPRDRV

Driver Option - Rise and fall times of the USB signal lines

10.5.3 Nested Class

```
public static final class FT_EEPROM_2232H.DRIVE_STRENGTH
extendsjava.lang.Object
```

The driver strength on the 2232H

Constructor Summary

Constructors

Constructor and Description
FT_EEPROM_2232H.DRIVE_STRENGTH()

10.5.4 Driver Length Fields

FT2232H Driver Strength	Constant variable (byte)	Constant Value
FT2232H Driver Strength 4mA	*DRIVER_4mA*	0
FT2232H Driver Strength 8mA	*DRIVER_8mA*	1
FT2232H Driver Strength 12mA	*DRIVER_12mA*	2
FT2232H Driver Strength 16mA	*DRIVER_16mA*	3

10.6 Class FT_EEPROM_4232H

public class **FT_EEPROM_4232H**
extends FT_EEPROM

EEPROM data structure on the 4232H

10.6.1 Constructor

Constructors

Constructor and Description
FT_EEPROM_4232H()

10.6.2Fields

Field Summary

Fields

Modifier and Type	Field and Description
byte	**AH_DriveCurrent** Drive Option - AH pins drive current.
boolean	**AH_LoadD2XX** Driver Option - Interface AH Load D2XX
boolean	**AH_LoadRI_RS485** Driver Option - Interface AH Load RI_RS485
boolean	**AH_LoadVCP** Driver Option - Interface AH Load Virtual Com Port
boolean	**AH_RI_TXDEN** Hardware Option - Interface AH uses RI as RS485 TXDEN
boolean	**AH_SchmittInput** Drive Option - AH pins have Schmitt input.
boolean	**AH_SlowSlew** Drive Option - AH pins have slow slew.
byte	**AL_DriveCurrent** Drive Option - AL pins drive current.
boolean	**AL_LoadD2XX** Driver Option - Interface AL Load D2XX
boolean	**AL_LoadRI_RS485** Driver Option - Interface AL Load RI_RS485
boolean	**AL_LoadVCP** Driver Option - Interface AL Load Virtual Com Port
boolean	**AL_RI_TXDEN** Hardware Option - Interface AL uses RI as RS485 TXDEN
boolean	**AL_SchmittInput** Drive Option - AL pins have Schmitt input.
boolean	**AL_SlowSlew** Drive Option - AL pins have slow slew.
byte	**BH_DriveCurrent** Drive Option - BH pins drive current.

boolean	**BH_LoadD2XX** Driver Option - Interface BH Load D2XX
boolean	**BH_LoadRI_RS485** Driver Option - Interface BH Load RI_RS485
boolean	**BH_LoadVCP** Driver Option - Interface BH Load Virtual Com Port
boolean	**BH_RI_TXDEN** Hardware Option - Interface BH uses RI as RS485 TXDEN
boolean	**BH_SchmittInput** Drive Option - BH pins have Schmitt input.
boolean	**BH_SlowSlew** Drive Option - BH pins have slow slew.
byte	**BL_DriveCurrent** Drive Option - BL pins Drive Current.
boolean	**BL_LoadD2XX** Driver Option - Interface BL Load D2XX
boolean	**BL_LoadRI_RS485** Driver Option - Interface BL Load RI_RS485
boolean	**BL_LoadVCP** Driver Option - Interface BL Load Virtual Com Port
boolean	**BL_RI_TXDEN** Hardware Option - Interface BL uses RI as RS485 TXDEN
boolean	**BL_SchmittInput** Drive Option - BL pins have Schmitt input.
boolean	**BL_SlowSlew** Drive Option - BL pins have slow slew.
int	**TPRDRV** Driver Option - Rise and fall times of the USB signal lines

Field Detail

AL_SlowSlew

public boolean AL_SlowSlew

Drive Option - AL pins have slow slew.

AL_SchmittInput

public boolean AL_SchmittInput

Drive Option - AL pins have Schmitt input.

AL_DriveCurrent

public byte AL_DriveCurrent

Drive Option - AL pins drive current.

AH_SlowSlew

public boolean AH_SlowSlew

Drive Option - AH pins have slow slew.

AH_SchmittInput

public boolean AH_SchmittInput

Drive Option - AH pins have Schmitt input.

AH_DriveCurrent

public byte AH_DriveCurrent

Drive Option - AH pins drive current.

BL_SlowSlew

Public boolean BL_SlowSlew

Drive Option - BL pins have slow slew.

BL_SchmittInput

public boolean BL_SchmittInput

Drive Option - BL pins have Schmitt input.

BL_DriveCurrent

public byte BL_DriveCurrent

Drive Option - BL pins Drive Current.

BH_SlowSlew

public boolean BH_SlowSlew

Drive Option - BH pins have slow slew.

BH_SchmittInput

public boolean BH_SchmittInput

Drive Option - BH pins have Schmitt input.

BH_DriveCurrent

public byte BH_DriveCurrent

Drive Option - BH pins drive current.

AL_RI_TXDEN

public boolean AL_RI_TXDEN

Hardware Option - Interface AL uses RI as RS485 TXDEN

AH_RI_TXDEN

public boolean AH_RI_TXDEN

Hardware Option - Interface AH uses RI as RS485 TXDEN

BL_RI_TXDEN

public boolean BL_RI_TXDEN

Hardware Option - Interface BL uses RI as RS485 TXDEN

BH_RI_TXDEN

publicboolean BH_RI_TXDEN

Hardware Option - Interface BH uses RI as RS485 TXDEN

AL_LoadVCP

public boolean AL_LoadVCP

Driver Option - Interface AL Load VirtualCom Port

AL_LoadD2XX

public boolean AL_LoadD2XX

Driver Option - Interface AL Load D2XX

AL_LoadRI_RS485

public boolean AL_LoadRI_RS485

Driver Option - Interface AL Load RI_RS485

AH_LoadVCP

public boolean AH_LoadVCP

Driver Option - Interface AH Load VirtualCom Port

AH_LoadD2XX

public boolean AH_LoadD2XX

Driver Option - Interface AH Load D2XX

AH_LoadRI_RS485

public boolean AH_LoadRI_RS485

Driver Option - Interface AH Load RI_RS485

BL_LoadVCP

public boolean BL_LoadVCP

Driver Option - Interface BL Load VirtualCom Port

BL_LoadD2XX

public boolean BL_LoadD2XX

Driver Option - Interface BL Load D2XX

BL_LoadRI_RS485

public boolean BL_LoadRI_RS485

Driver Option - Interface BL Load RI_RS485

BH_LoadVCP

public boolean BH_LoadVCP

Driver Option - Interface BH Load VirtualCom Port

BH_LoadD2XX

public boolean BH_LoadD2XX

Driver Option - Interface BH Load D2XX

BH_LoadRI_RS485

public boolean BH_LoadRI_RS485

Driver Option - Interface BH Load RI_RS485

TPRDRV

public int TPRDRV

Driver Option - fluctuating times of the USB signal lines

10.6.3Nested Class

public static final class **FT_EEPROM_4232H.DRIVE_STRENGTH**
extends java.lang.Object

The driver strength on the FT4232H

Constructor Summary

Constructors

Constructor and Description
FT_EEPROM_4232H.DRIVE_STRENGTH()

10.6.4 Driver Length Fields

FT4232H Driver Strength	Constant variable (byte)	Constant Value
FT4232H Driver Strength 4mA	*DRIVER_4mA*	0
FT4232H Driver Strength 8mA	*DRIVER_8mA*	1
FT4232H Driver Strength 12mA	*DRIVER_12mA*	2
FT4232H Driver Strength 16mA	*DRIVER_16mA*	3

10.7 Class FT_EEPROM_232H

```
public class FT_EEPROM_232H
extends FT_EEPROM
```

EEPROM data structure on the 232H

10.7.1 Constructor

Constructors

Constructor and Description
FT_EEPROM_232H()

10.7.2Fields

Field Summary

Fields

Modifier and Type	Field and Description
byte	**AL_DriveCurrent** Drive Option - AL pins drive current.
boolean	**AL_SchmittInput** Drive Option - AL pins have Schmitt input.
boolean	**AL_SlowSlew** Drive Option - AL pins have slow slew.
byte	**BL_DriveCurrent** Drive Option - BL pins drive current.
boolean	**BL_SchmittInput** Drive Option - BL pins have Schmitt input.
boolean	**BL_SlowSlew** Drive Option - BL pins have slow slew.
byte	**CBus0** Hardware Option - CBus0 Mux Control
byte	**CBus1** Hardware Option - CBus1 Mux Control
byte	**CBus2** Hardware Option - CBus2 Mux Control
byte	**CBus3** Hardware Option - CBus3 Mux Control
byte	**CBus4** Hardware Option - CBus4 Mux Control
byte	**CBus5** Hardware Option - CBus5 Mux Control
byte	**CBus6** Hardware Option - CBus6 Mux Control
byte	**CBus7** Hardware Option - CBus7 Mux Control
byte	**CBus8**

	Hardware Option - CBus8 Mux Control
byte	**CBus9** Hardware Option - CBus9 Mux Control
boolean	**FastSerial** Hardware Option - Fast Serial
boolean	**FIFO** Hardware Option - 245 FIFO
boolean	**FIFOTarget** Hardware Option - 245 FIFO CPU Target
boolean	**FT1248** Hardware Option - FT1248
boolean	**FT1248ClockPolarity** FT1248 Option - FT1248 clock polarity, true = clock idle high, false = clock idle low
boolean	**FT1248FlowControl** FT1248 Option - FT1248 Flow Control
boolean	**FT1248LSB** FT1248 Option - FT1248 LSB, true = LSB, false = MSB
boolean	**LoadD2XX** Driver Option - Load D2XX
boolean	**LoadVCP** Driver Option - Load Virtual Com Port
boolean	**PowerSaveEnable** Hardware Option - Power Save Enable if using BCBus7 to save power for self-powered designs.
boolean	**UART** Hardware Option - UART

Field Detail

AL_SlowSlew

public boolean AL_SlowSlew

Drive Option - AL pins have slow slew.

AL_SchmittInput

public boolean AL_SchmittInput

Drive Option - AL pins have Schmitt input.

AL_DriveCurrent

public byte AL_DriveCurrent

Drive Option - AL pins drive current.

BL_SlowSlew

public boolean BL_SlowSlew

Drive Option - BL pins have slow slew.

BL_SchmittInput

public boolean BL_SchmittInput

Drive Option - BL pins have Schmitt input.

BL_DriveCurrent

public byte BL_DriveCurrent

Drive Option - BL pins drive current.

CBus0

public byte CBus0

Hardware Option - CBus0 Mux Control

CBus1

public byte CBus1

Hardware Option - CBus1 Mux Control

CBus2

public byte CBus2

Hardware Option - CBus2 Mux Control

CBus3

public byte CBus3

Hardware Option - CBus3 Mux Control

CBus4

public byte CBus4

Hardware Option - CBus4 Mux Control

CBus5

public byte CBus5

Hardware Option - CBus5 Mux Control

CBus6

public byte CBus6

Hardware Option - CBus6 Mux Control

CBus7

public byte CBus7

Hardware Option - CBus7 Mux Control

CBus8

public byte CBus8

Hardware Option - CBus8 Mux Control

CBus9

public byte CBus9

Hardware Option - CBus9 Mux Control

UART

public boolean UART

Hardware Option - UART

FIFO

public boolean FIFO

Hardware Option - 245 FIFO

FIFOTarget

public boolean FIFOTarget

Hardware Option - 245 FIFO CPU Target

FastSerial

public boolean FastSerial

Hardware Option - Fast Serial

FT1248

public boolean FT1248

Hardware Option - FT1248

FT1248ClockPolarity

public boolean FT1248ClockPolarity

FT1248 Option - FT1248 clock polarity, true = clock idle high, false = clock idle low

FT1248LSB

public boolean FT1248LSB

FT1248 Option - FT1248 LSB, true = LSB, false = MSB

FT1248FlowControl

public boolean FT1248FlowControl

FT1248 Option - FT1248 Flow Control

PowerSaveEnable

public boolean PowerSaveEnable

Hardware Option - Power Save Enable if using BCBus7 to save power for self-powered designs.

LoadVCP

public boolean LoadVCP

Driver Option - Load Virtual Com Port

LoadD2XX

public boolean LoadD2XX

Driver Option - Load D2XX

10.7.3Nested Class – Driver Strength

public static final class **FT_EEPROM_232H.DRIVE_STRENGTH**
extends java.lang.Object

The driver strength on the 232H

Constructor Summary

Constructors
Constructor and Description
FT_EEPROM_232H.DRIVE_STRENGTH()

10.7.4 Driver Length Fields

FT232H Driver Strength	Constant variable (byte)	Constant Value
FT232H Driver Strength 4mA	*DRIVER_4mA*	0
FT232H Driver Strength 8mA	*DRIVER_8mA*	1
FT232H Driver Strength 12mA	*DRIVER_12mA*	2
FT232H Driver Strength 16mA	*DRIVER_16mA*	3

10.7.5 Nested Class – CBBUS

public static final class **FT_EEPROM_232H.CBUS**
extends java.lang.Object

CBus Option on the FT232H

Constructor Summary

Constructors
Constructor and Description
FT_EEPROM_232H.CBUS()

10.7.6 CBUS Fields

FT 232H CBUS Option	CBUS Constant Variable (int)	Constant Value
Tri State #	TRISTATE	0
PWRON #	PWRON	1
RXLED #	RXLED	2
TXLED #	TXLED	3
Power Enable #	PWREN	4
SLEEP #	SLEEP	5
Driver 0 #	DRIVER_0	6
Driver 1 #	DRIVER_1	7
GPIO Mode #	GPIO_MODE	8
TXDEN #	TXDEN	9
30M Hz Clock Output #	CLK30MHz	10
15M HzClock Output #	CLK15MHz	11
7.5M Hz Clock Output #	CLK7_5MHz	12

10.8 Class FT_EEPROM_X_Series

public class **FT_EEPROM_X_Series**
extends FT_EEPROM

EEPROM data structure on the X Series

10.8.1 Constructor

Constructors

Constructor and Description
FT_EEPROM_X_Series()

10.8.2Fields

Field Summary

Fields

Modifier and Type	Field and Description
short	**A_DeviceTypeValue**
boolean	**A_LoadD2XX** Driver Option - Load D2XX
boolean	**A_LoadVCP** Driver Option - Load Virtual Com Port
byte	**AC_DriveCurrent** Drive Option - AC pins drive current.
boolean	**AC_SchmittInput** Drive Option - AC pins have Schmitt input.
boolean	**AC_SlowSlew** Drive Option - AC pins have slow slew.
byte	**AD_DriveCurrent** Drive Option - AD pins drive current.
boolean	**AD_SchmittInput** Drive Option - AD pins have Schmitt input.
boolean	**AD_SlowSlew** Drive Option - AD pins have slow slew.
boolean	**BCDDisableSleep** Battery Charge Detect option - Disable Sleep
boolean	**BCDEnable** Battery Charge Detect option - Enable
boolean	**BCDForceCBusPWREN** Battery Charge Detect option - Force CBus Power Enable
byte	**CBus0** Hardware Option - CBus0 Mux Control
byte	**CBus1** Hardware Option - CBus1 Mux Control
byte	**CBus2** Hardware Option - CBus2 Mux Control
byte	**CBus3**

	Hardware Option - CBus3 Mux Control
byte	**CBus4** Hardware Option - CBus4 Mux Control
byte	**CBus5** Hardware Option - CBus5 Mux Control
byte	**CBus6** Hardware Option - CBus6 Mux Control
boolean	**FT1248ClockPolarity** FT1248 Option - FT1248 clock polarity, true = clock idle high, false = clock idle low
boolean	**FT1248FlowControl** FT1248 Option - FT1248 Flow Control
boolean	**FT1248LSB** FT1248 Option - FT1248 LSB, true = LSB, false = MSB
int	**I2CDeviceID** I2C Option - I2C Device ID
boolean	**I2CDisableSchmitt** I2C Option - Disable Schmitt trigger
int	**I2CSlaveAddress** I2C Option - Slave Address
boolean	**InvertCTS** Hardware Option - Invert CTS signal
boolean	**InvertDCD** Hardware Option - Invert DCD signal
boolean	**InvertDSR** Hardware Option - Invert DSR signal
boolean	**InvertDTR** Hardware Option - Invert DTR signal
boolean	**InvertRI** Hardware Option - Invert RI signal
boolean	**InvertRTS** Hardware Option - Invert RTS signal
boolean	**InvertRXD** Hardware Option - Invert RXD signal
boolean	**InvertTXD** Hardware Option - Invert TXD signal
boolean	**PowerSaveEnable** Hardware Option - Power Save Enable if using BCBus7 to save power for self-powered designs.
boolean	**RS485EchoSuppress** Hardware Option - RS485 Echo Suppression

Field Detail

A_DeviceTypeValue

public short A_DeviceTypeValue

A_LoadVCP

public boolean A_LoadVCP

Driver Option - Load Virtual Com Port

A_LoadD2XX

public boolean A_LoadD2XX

Driver Option - Load D2XX

BCDEnable

public boolean BCDEnable

Battery Charge Detect option - Enable

BCDForceCBusPWREN

public boolean BCDForceCBusPWREN

Battery Charge Detect option - Force CBus Power Enable

BCDDisableSleep

public boolean BCDDisableSleep

Battery Charge Detect option - Disable Sleep

CBus0

public byte CBus0

Hardware Option - CBus0 Mux Control

CBus1

public byte CBus1

Hardware Option - CBus1 Mux Control

CBus2

public byte CBus2

Hardware Option - CBus2 Mux Control

CBus3

public byte CBus3

Hardware Option - CBus3 Mux Control

CBus4

public byte CBus4

Hardware Option - CBus4 Mux Control

CBus5

public byte CBus5

Hardware Option - CBus5 Mux Control

CBus6

public byte CBus6

Hardware Option - CBus6 Mux Control

FT1248ClockPolarity

public boolean FT1248ClockPolarity

FT1248 Option - FT1248 clock polarity, true = clock idle high, false = clock idle low

FT1248LSB

public boolean FT1248LSB

FT1248 Option - FT1248 LSB, true = LSB, false = MSB

FT1248FlowControl

public boolean FT1248FlowControl

FT1248 Option - FT1248 Flow Control

InvertTXD

public boolean InvertTXD

Hardware Option - Invert TXD signal

InvertRXD

public boolean InvertRXD

Hardware Option - Invert RXD signal

InvertRTS

public boolean InvertRTS

Hardware Option - Invert RTS signal

InvertCTS

public boolean InvertCTS

Hardware Option - Invert CTS signal

InvertDTR

public boolean InvertDTR

Hardware Option - Invert DTR signal

InvertDSR

public boolean InvertDSR

Hardware Option - Invert DSR signal

InvertDCD

public boolean InvertDCD

Hardware Option - Invert DCD signal

InvertRI

public boolean InvertRI

Hardware Option - Invert RI signal

I2CSlaveAddress

public int I2CSlaveAddress

I2C Option - Slave Address

I2CDeviceID

public int I2CDeviceID

I2C Option - I2C Device ID

I2CDisableSchmitt

public boolean I2CDisableSchmitt

I2C Option - Disable Schmitt trigger

AD_SlowSlew

public boolean AD_SlowSlew

Drive Option - AD pins have slow slew.

AD_SchmittInput

public boolean AD_SchmittInput

Drive Option - AD pins have Schmitt input.

AD_DriveCurrent

public byte AD_DriveCurrent

Drive Option - AD pins drive current.

AC_SlowSlew

public boolean AC_SlowSlew

Drive Option - AC pins have slow slew.

AC_SchmittInput

public boolean AC_SchmittInput

Drive Option - AC pins have Schmitt input.

AC_DriveCurrent

public byte AC_DriveCurrent

Drive Option - AC pins drive current.

RS485EchoSuppress

public boolean RS485EchoSuppress

Hardware Option - RS485 Echo Suppression

PowerSaveEnable

public boolean PowerSaveEnable

Hardware Option - Power Save Enable if using BCBus7 to save power for self-powered designs.

10.8.3 Nested Class – Driver Strength

public static final class FT_EEPROM_X_Series.DRIVE_STRENGTH
extends java.lang.Object

The driver strength on the X Series

10.8.4 Driver Length Fields

FT X Series Driver Strength	Constant variable (byte)	Constant Value
FT X Series Driver Strength 4mA	*DRIVER_4mA*	0
FT X Series Driver Strength 8mA	*DRIVER_8mA*	1
FT X Series Driver Strength 12mA	*DRIVER_12mA*	2
FT X Series Driver Strength 16mA	*DRIVER_16mA*	3

10.8.5 Nested Class – CBUS

public static final class FT_EEPROM_X_Series.CBUS
extends java.lang.Object

CBUS Option on the X Series

10.8.6 CBUS Fields

FT X SeriesCBUS Option	CBUS Constant Variable (int)	Constant Value
Tri State #	TRISTATE	0
RXLED #	RXLED	1
TXLED #	TXLED	2
TX & RX LED #	TXRXLED	3
Power Enable #	PWREN	4
SLEEP #	SLEEP	5
Driver 0 #	DRIVER_0	6
Driver 1 #	DRIVER_1	7
GPIO Mode #	GPIO_MODE	8
TXDEN #	TXDEN	9
24M Hz Clock Output #	CLK24MHz	10
12M Hz Clock Output #	CLK12MHz	11
6M Hz Clock Output #	CLK6MHz	12
BCD Charge 1 #	BCD_Charge1	13
BCDCharge2 #	BCD_Charge2	14
I2C TXE #	I2C_TXE	15
I2C RXF #	I2C_RXF	16
VBUS Sense #	VBUS_Sense	17
Bit Bang WR #	BitBang_WR	18
Bit Bang RD #	BitBang_RD	19
Time Stamp #	Time_Stamp	20
Keep Awake #	Keep_Awake	21

11 Appendix A – References

http://developer.Android.com/index.html

http://www.ftdichip.com/

12 Appendix B – List of figures

13 Appendix C – Revision History

Document Title:	Android D2xx API User Manual
Document Reference No.:	FT_000796
Clearance No.:	FTDI# 328
Drivers Page:	http://www.ftdichip.com/Drivers/D2XX.htm
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial Release for beta test	2011-01-04

14 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G411HH
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
Web Site URL	http://www.ftdichip.com
Web Shop URL	http://www.ftdichip.com

Branch Office – Taipei, Taiwan

2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 87913570
Fax: +886 (0) 2 8791 3576

E-mail (Sales)	tw.sales1@ftdichip.com
E-mail (Support)	tw.support1@ftdichip.com
E-mail (General Enquiries)	tw.admin1@ftdichip.com
Web Site URL	http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, 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
Web Site URL	http://www.ftdichip.com

Branch Office – Shanghai, China

Room 408, 317Xianxia Road,
Shanghai, 200051
China
Tel: +86 2162351596
Fax: +86 2162351595

E-mail (Sales)	cn.sales@ftdichip.com
E-mail (Support)	cn.support@ftdichip.com
E-mail (General Enquiries)	cn.admin@ftdichip.com
Web Site URL	http://www.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.