

**D2XX Programmer's Guide**

### Version 1.5

#### Issue Date: 07-09-2023

FTDI provides DLL and virtual COM port (VCP) application interfaces to its drivers. This document provides the application programming interface (API) for the FTD2XX DLL function library.

Use of FTDI devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold FTDI harmless from any and all damages, claims, suits or expense resulting from such use.

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

Web Site: [http://ftdichip.com](http://ftdichip.com/)

Copyright © Future Technology Devices International Limited

**Table of Contents**

1. [Preface 5](#_bookmark0)
2. [Introduction 6](#_bookmark1)
3. [D2XX Classic Functions 7](#_bookmark3)
   1. [FT\_SetVIDPID 7](#_bookmark4)
   2. [FT\_GetVIDPID 7](#_bookmark5)
   3. [FT\_CreateDeviceInfoList 8](#_bookmark6)
   4. [FT\_GetDeviceInfoList 9](#_bookmark7)
   5. [FT\_GetDeviceInfoDetail 10](#_bookmark8)
   6. [FT\_ListDevices 12](#_bookmark9)
   7. [FT\_Open 14](#_bookmark10)
   8. [FT\_OpenEx 15](#_bookmark11)
   9. [FT\_Close 18](#_bookmark12)
   10. [FT\_Read 18](#_bookmark13)
   11. [FT\_Write 20](#_bookmark14)
   12. [FT\_SetBaudRate 21](#_bookmark15)
   13. [FT\_SetDivisor 22](#_bookmark16)
   14. [FT\_SetDataCharacteristics 22](#_bookmark17)
   15. [FT\_SetTimeouts 23](#_bookmark18)
   16. [FT\_SetFlowControl 24](#_bookmark19)
   17. [FT\_SetDtr 25](#_bookmark20)
   18. [FT\_ClrDtr 26](#_bookmark21)
   19. [FT\_SetRts 27](#_bookmark22)
   20. [FT\_ClrRts 28](#_bookmark23)
   21. [FT\_GetModemStatus 28](#_bookmark24)
   22. [FT\_GetQueueStatus 29](#_bookmark26)
   23. [FT\_GetDeviceInfo 30](#_bookmark27)
   24. [FT\_GetDriverVersion 32](#_bookmark28)
   25. [FT\_GetLibraryVersion 33](#_bookmark29)
   26. [FT\_GetComPortNumber 33](#_bookmark30)
   27. [FT\_GetStatus 34](#_bookmark31)
   28. [FT\_SetEventNotification 35](#_bookmark32)
   29. [FT\_SetChars 37](#_bookmark33)
   30. [FT\_SetBreakOn 38](#_bookmark34)
   31. [FT\_SetBreakOff 39](#_bookmark35)
   32. [FT\_Purge 39](#_bookmark36)
   33. [FT\_ResetDevice 40](#_bookmark37)
   34. [FT\_ResetPort 41](#_bookmark38)
   35. [FT\_CyclePort 41](#_bookmark39)
   36. [FT\_Rescan 42](#_bookmark40)
   37. [FT\_Reload 43](#_bookmark41)
   38. [FT\_SetResetPipeRetryCount 44](#_bookmark42)
   39. [FT\_StopInTask 44](#_bookmark43)
   40. [FT\_RestartInTask 45](#_bookmark44)
   41. [FT\_SetDeadmanTimeout 46](#_bookmark45)
   42. [FT\_IoCtl 47](#_bookmark46)
   43. [FT\_SetWaitMask 47](#_bookmark47)
   44. [FT\_WaitOnMask 47](#_bookmark48)
4. [EEPROM Programming Interface Functions 48](#_bookmark49)
   1. [FT\_ReadEE 48](#_bookmark50)
   2. [FT\_WriteEE 49](#_bookmark51)
   3. [FT\_EraseEE 49](#_bookmark52)
   4. [FT\_EE\_Read 50](#_bookmark53)
   5. [FT\_EE\_ReadEx 51](#_bookmark54)
   6. [FT\_EE\_Program 52](#_bookmark55)
   7. [FT\_EE\_ProgramEx 55](#_bookmark56)
   8. [FT\_EE\_UASize 56](#_bookmark57)
   9. [FT\_EE\_UARead 57](#_bookmark58)
   10. [FT\_EE\_UAWrite 58](#_bookmark59)
   11. [FT\_EEPROM\_Read 59](#_bookmark60)
   12. [FT\_EEPROM\_Program 61](#_bookmark61)
5. [Extended API Functions 64](#_bookmark62)
   1. [FT\_SetLatencyTimer 64](#_bookmark63)
   2. [FT\_GetLatencyTimer 65](#_bookmark64)
   3. [FT\_SetBitMode 66](#_bookmark65)
   4. [FT\_GetBitMode 67](#_bookmark66)
   5. [FT\_SetUSBParameters 68](#_bookmark67)
6. [FT-Win32 API Functions 70](#_bookmark68)
   1. [FT\_W32\_CreateFile 70](#_bookmark69)
   2. [FT\_W32\_CloseHandle 72](#_bookmark70)
   3. [FT\_W32\_ReadFile 72](#_bookmark71)
   4. [FT\_W32\_WriteFile 75](#_bookmark72)
   5. [FT\_W32\_GetOverlappedResult 77](#_bookmark73)
   6. [FT\_W32\_EscapeCommFunction 77](#_bookmark74)
   7. [FT\_W32\_GetCommModemStatus 78](#_bookmark75)
   8. [FT\_W32\_SetupComm 79](#_bookmark76)
   9. [FT\_W32\_SetCommState 80](#_bookmark77)
   10. [FT\_W32\_GetCommState 80](#_bookmark78)
   11. [FT\_W32\_SetCommTimeouts 81](#_bookmark79)
   12. [FT\_W32\_GetCommTimeouts 82](#_bookmark80)
   13. [FT\_W32\_SetCommBreak 83](#_bookmark81)
   14. [FT\_W32\_ClearCommBreak 84](#_bookmark82)
   15. [FT\_W32\_SetCommMask 84](#_bookmark83)
   16. [FT\_W32\_GetCommMask 85](#_bookmark84)
   17. [FT\_W32\_WaitCommEvent 86](#_bookmark85)
   18. [FT\_W32\_PurgeComm 88](#_bookmark86)
   19. [FT\_W32\_GetLastError 89](#_bookmark87)
   20. [FT\_W32\_ClearCommError 89](#_bookmark88)
7. [Contact Information 91](#_bookmark89)

[Appendix A - Type Definitions 92](#_bookmark90)

[Appendix B – References 106](#_bookmark104)

[Document References 106](#_bookmark105)

[Acronyms and Abbreviations 106](#_bookmark106)

[Appendix C – List of Tables & Figures 107](#_bookmark107)

[List of Tables 107](#_bookmark108)

[List of Figures 107](#_bookmark109)

[Appendix D - Revision History 108](#_bookmark110)

# Preface

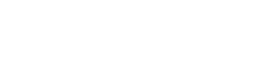
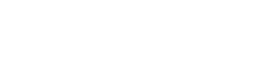
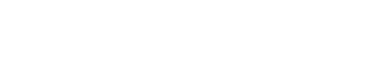
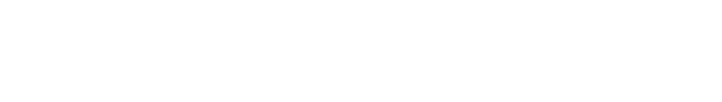
The D2XX interface is a proprietary interface specifically for FTDI devices. This document provides an explanation of the functions available to application developers via the FTD2XX library.

Any software code examples given in this document are for information only. The examples are not guaranteed and are not supported by FTDI.

# Introduction

FTDI provides two alternative software interfaces for its range of USB-UART and USB-FIFO ICs. One interface provides a Virtual COM Port (VCP) which appears to the system as a legacy COM port. The second interface, D2XX, is provided via a proprietary DLL (FTD2XX.DLL). The D2XX interface provides special functions that are not available in standard operating system COM port APIs, such as setting the device into a different mode or writing data into the device EEPROM.

In the case of the FTDI drivers for Windows, the D2XX driver and VCP driver are distributed in the same driver package, called the Combined Driver Model (CDM) package. [Figure 2.1 Windows CDM Driver](#_bookmark2) [Architecture](#_bookmark2) illustrates the architecture of the Windows CDM driver.



**D2XX Application**

**COM Port Application**

**FTD2XX.DLL**

**FTSER2K.SYS**

**FTDIBUS.SYS**

**USB Host Controller Driver**

#### Figure 2.1 Windows CDM Driver Architecture

For Linux, Mac OS X (10.4 and later) and Windows CE (4.2 and later) the D2XX driver and VCP driver are mutually exclusive options as only one driver type may be installed at a given time for a given device ID. In the case of a Windows system running the CDM driver, applications may use either the D2XX or VCP interface without installing a different driver but may not use both interfaces at the same time.

As the VCP driver interface is designed to emulate a legacy COM port, FTDI does not provide documentation on how to communicate with the VCP driver from an application; the developer is referred to the large amount of material available on the Internet regarding serial communication.

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 FTD2XX library.

# D2XX Classic Functions

The functions listed in this section are compatible with all FTDI devices.

## FT\_SetVIDPID

Supported Operating Systems Linux

Mac OS X (10.4 and later)

### Summary

A command to include a custom VID and PID combination within the internal device list table. This will allow the driver to load for the specified VID and PID combination.

### Definition

FT\_STATUS **FT\_SetVIDPID** (DWORD *dwVID*, DWORD *dwPID*)

### Parameters

*dwVID* Device Vendor ID (VID)

*dwPID* Device Product ID (PID)

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

By default, the driver will support a limited set of VID and PID matched devices (VID 0x0403 with PIDs 0x6001, 0x6010, 0x6006 only).

In order to use the driver with other VID and PID combinations the FT\_SetVIDPID function must be used prior to calling [FT\_ListDevices](#_bookmark9), [FT\_Open](#_bookmark10), [FT\_OpenEx](#_bookmark11) or [FT\_CreateDeviceInfoList](#_bookmark6).

## FT\_GetVIDPID

### Supported Operating Systems

Linux

Mac OS X (10.4 and later)

### Summary

A command to retrieve the current VID and PID combination from within the internal device list table.

### Definition

FT\_STATUS **FT\_GetVIDPID** (DWORD \* *pdwVID*, DWORD \* *pdwPID*)

### Parameters

*pdwVID* Pointer to DWORD that will contain the internal VID

*pdwPID* Pointer to DWORD that will contain the internal PID

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

See [FT\_SetVIDPID](#_bookmark4).

## FT\_CreateDeviceInfoList

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

This function builds a device information list and returns the number of D2XX devices connected to the system. The list contains information about both unopen and open devices.

### Definition

FT\_STATUS **FT\_CreateDeviceInfoList** (LPDWORD *lpdwNumDevs*)

### Parameters

*lpdwNumDevs* Pointer to unsigned long to store the number of devices connected.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

An application can use this function to get the number of devices attached to the system. It can then allocate space for the device information list and retrieve the list using [FT\_GetDeviceInfoList](#_bookmark7) or [FT\_GetDeviceInfoDetailFT\_GetDeviceInfoDetail](#_bookmark8).

If the devices connected to the system change, the device info list will not be updated until FT\_CreateDeviceInfoList is called again.

### Example

FT\_STATUS ftStatus;

DWORD numDevs;

// create the device information list ftStatus = FT\_CreateDeviceInfoList(&numDevs); if (ftStatus == FT\_OK) {

printf("Number of devices is %d\n",numDevs);

}

else {

}

// FT\_CreateDeviceInfoList failed

## FT\_GetDeviceInfoList

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function returns a device information list and the number of D2XX devices in the list.

### Definition

FT\_STATUS **FT\_GetDeviceInfoList** (FT\_DEVICE\_LIST\_INFO\_NODE \**pDest*,

LPDWORD *lpdwNumDevs*)

### Parameters

*\*pDest* Pointer to an array of [FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102) structures.

*lpdwNumDevs* Pointer to the number of elements in the array.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function should only be called after calling [FT\_CreateDeviceInfoList](#_bookmark6). If the devices connected to the system change, the device info list will not be updated until [FT\_CreateDeviceInfoList](#_bookmark6) is called again.

Location ID information is not returned for devices that are open when [FT\_CreateDeviceInfoList](#_bookmark6) is called. Information is not available for devices which are open in other processes. In this case, the *Flags*

parameter of the [FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102) will indicate that the device is open, but other fields will

be unpopulated.

The flag value is a 4-byte bit map containing miscellaneous data as defined in [Appendix A – Type](#_bookmark103) [Definitions](#_bookmark103). Bit 0 (least significant bit) of this number indicates if the port is open (1) or closed (0). Bit 1 indicates if the device is enumerated as a high-speed USB device (2) or a full-speed USB device (0). The remaining bits (2 - 31) are reserved.

The array of [FT\_DEVICE\_LIST\_INFO\_NODES](#_bookmark102) contains all available data on each device. The structure of [FT\_DEVICE\_LIST\_INFO\_NODES](#_bookmark102) is given in the Appendix. The storage for the list must be allocated by the application. The number of devices returned by [FT\_CreateDeviceInfoList](#_bookmark6) can be used to do this.

When programming in Visual Basic, LabVIEW or similar languages, [FT\_GetDeviceInfoDetail](#_bookmark8) may be required instead of this function.

Please note that Linux, Mac OS X and Windows CE do not support location IDs. As such, the Location ID parameter in the structure will be empty under these operating systems.

#### Example

FT\_STATUS ftStatus; FT\_DEVICE\_LIST\_INFO\_NODE \*devInfo; DWORD numDevs;

// create the device information list ftStatus = FT\_CreateDeviceInfoList(&numDevs);

if (ftStatus == FT\_OK) {

printf("Number of devices is %d\n",numDevs);

}

if (numDevs > 0) {

// allocate storage for list based on numDevs devInfo =

(FT\_DEVICE\_LIST\_INFO\_NODE\*)malloc(sizeof(FT\_DEVICE\_LIST\_INFO\_NODE)\*numDevs);

// get the device information list

ftStatus = FT\_GetDeviceInfoList(devInfo,&numDevs); if (ftStatus == FT\_OK) {

for (int i = 0; i < numDevs; i++) { printf("Dev %d:\n",i);

printf(" Flags=0x%x\n",devInfo[i].Flags); printf(" Type=0x%x\n",devInfo[i].Type); printf(" ID=0x%x\n",devInfo[i].ID); printf(" LocId=0x%x\n",devInfo[i].LocId);

printf(" SerialNumber=%s\n",devInfo[i].SerialNumber); printf(" Description=%s\n",devInfo[i].Description); printf(" ftHandle=0x%x\n",devInfo[i].ftHandle);

}

}

}

## FT\_GetDeviceInfoDetail

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function returns an entry from the device information list.

### Definition

FT\_STATUS **FT\_GetDeviceInfoDetail** (DWORD *dwIndex*, LPDWORD *lpdwFlags*,

LPDWORD *lpdwType*,

LPDWORD *lpdwID*, LPDWORD *lpdwLocId*, PCHAR pcSerialNumber, PCHAR pcDescription, FT\_HANDLE *\*ftHandle*)

### Parameters

*dwIndex* Index of the entry in the device info list.

*lpdwFlags* Pointer to unsigned long to store the flag value.

*lpdwType* Pointer to unsigned long to store device type.

*lpdwID* Pointer to unsigned long to store device ID.

*lpdwLocId* Pointer to unsigned long to store the device location ID.

*pcSerialNumber* Pointer to buffer to store device serial number as a null-

terminated string.

*pcDescription* Pointer to buffer to store device description as a null-terminated

string.

*\*ftHandle* Pointer to a variable of type FT\_HANDLE where the handle will be stored.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function should only be called after calling [FT\_CreateDeviceInfoList](#_bookmark6). If the devices connected to the system change, the device info list will not be updated until [FT\_CreateDeviceInfoList](#_bookmark6) is called again.

The index value is zero-based.

The flag value is a 4-byte bit map containing miscellaneous data as defined in [Appendix A – Type](#_bookmark103) [Definitions](#_bookmark103). Bit 0 (least significant bit) of this number indicates if the port is open (1) or closed (0). Bit 1 indicates if the device is enumerated as a high-speed USB device (2) or a full-speed USB device (0). The remaining bits (2 - 31) are reserved.

Location ID information is not returned for devices that are open when [FT\_CreateDeviceInfoList](#_bookmark6) is called. Information is not available for devices which are open in other processes. In this case, the *lpdwFlags*

parameter will indicate that the device is open, but other fields will be unpopulated.

To return the whole device info list as an array of [FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102) structures, use [FT\_CreateDeviceInfoList](#_bookmark6).

Please note that Linux, Mac OS X and Windows CE do not support location IDs. As such, the Location ID parameter in the structure will be empty under these operating systems.

### Example

FT\_STATUS ftStatus;

FT\_HANDLE ftHandleTemp;

DWORD numDevs;

DWORD Flags;

DWORD ID;

DWORD Type;

DWORD LocId;

char SerialNumber[16]; char Description[64];

// create the device information list ftStatus = FT\_CreateDeviceInfoList(&numDevs); if (ftStatus == FT\_OK) {

printf("Number of devices is %d\n",numDevs);

}

if (numDevs > 0) {

// get information for device 0

ftStatus = FT\_GetDeviceInfoDetail(0, &Flags, &Type, &ID, &LocId, SerialNumber, Description, &ftHandleTemp);

if (ftStatus == FT\_OK) { printf("Dev 0:\n");

printf(" Flags=0x%x\n",Flags); printf(" Type=0x%x\n",Type); printf(" ID=0x%x\n",ID); printf(" LocId=0x%x\n",LocId);

printf(" SerialNumber=%s\n",SerialNumber); printf(" Description=%s\n",Description); printf(" ftHandle=0x%x\n",ftHandleTemp);

}

}

## FT\_ListDevices

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Gets information concerning the devices currently connected. This function can return information such

as the number of devices connected, the device serial number and device description strings, and the location IDs of connected devices.

### Definition

FT\_STATUS **FT\_ListDevices** (PVOID *pvArg1*, PVOID *pvArg2*, DWORD *dwFlags*)

### Parameters

*pvArg1* Meaning depends on dwFlags.

*pvArg2* Meaning depends on dwFlags.

*dwFlags* Determines format of returned information.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function can be used in a number of ways to return different types of information. A more powerful way to get device information is to use the [FT\_CreateDeviceInfoList](#_bookmark6), [FT\_GetDeviceInfoList](#_bookmark7) and [FT\_GetDeviceInfoDetail](#_bookmark8) functions as they return all the available information on devices.

In its simplest form, it can be used to return the number of devices currently connected. If [*FT\_LIST\_NUMBER\_ONLY*](#_bookmark92) bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to a DWORD location to store the number of devices currently connected.

It can be used to return device information: if [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93) bit is set in *dwFlags*, the serial number string will

be returned; if [*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) bit is set in *dwFlags*, the product description string will be returned; if [*FT\_OPEN\_BY\_LOCATION*](#_bookmark93) bit is set in *dwFlags*, the Location ID will be returned; if none of these bits is set, the serial number string will be returned by default.

It can be used to return device string information for a single device. If [*FT\_LIST\_BY\_INDEX*](#_bookmark92) and [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93) or [*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) bits are set in dwFlags, the parameter pvArg1 is interpreted as the index of the device, and the parameter pvArg2 is interpreted as a pointer to a buffer to contain the appropriate string. Indexes are zero-based, and the error code [*FT\_DEVICE\_NOT\_FOUND*](#_bookmark91) is returned for an invalid index.

It can be used to return device string information for all connected devices. If [*FT\_LIST\_ALL*](#_bookmark92)and [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93) or [*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) bits are set in dwFlags, the parameter pvArg1 is interpreted as a pointer to an array of pointers to buffers to contain the appropriate strings and the parameter pvArg2 is interpreted as a pointer to a DWORD location to store the number of devices currently connected. Note that, for pvArg1, the last entry in the array of pointers to buffers should be a NULL pointer so the array will contain one more location than the number of devices connected.

The location ID of a device is returned if [*FT\_LIST\_BY\_INDEX*](#_bookmark92) and [*FT\_OPEN\_BY\_LOCATION*](#_bookmark93) bits are set in *dwFlags*. In this case the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a variable of type long to contain the location ID. Indexes are zero- based, and the error code [*FT\_DEVICE\_NOT\_FOUND*](#_bookmark91) is returned for an invalid index. Please note that Windows CE and Linux do not support location IDs.

The location IDs of all connected devices are returned if [*FT\_LIST\_ALL*](#_bookmark92) and [*FT\_OPEN\_BY\_LOCATION*](#_bookmark93) bits are set in *dwFlags*. In this case, the parameter *pvArg1* is interpreted as a pointer to an array of variables of type long to contain the location IDs, and the parameter *pvArg2* is interpreted as a pointer to a DWORD location to store the number of devices currently connected.

### Examples

The examples that follow use these variables.

FT\_STATUS ftStatus;

DWORD numDevs;

1. Get the number of devices currently connected

ftStatus = FT\_ListDevices(&numDevs,NULL,FT\_LIST\_NUMBER\_ONLY); if (ftStatus == FT\_OK) {

// FT\_ListDevices OK, number of devices connected is in numDevs

}

else {

}

// FT\_ListDevices failed

1. Get serial number of first device

DWORD devIndex = 0; // first device

char Buffer[64]; // more than enough room!

ftStatus = FT\_ListDevices((PVOID)devIndex,Buffer,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_SERIAL\_NUMBER); if (ftStatus == FT\_OK) {

// FT\_ListDevices OK, serial number is in Buffer

}

else {

}

// FT\_ListDevices failed

Note that indexes are zero-based. If more than one device is connected, incrementing *devIndex* will get the serial number of each connected device in turn.

1. Get device descriptions of all devices currently connected

char \*BufPtrs[3]; // pointer to array of 3 pointers

char Buffer1[64]; // buffer for description of first device char Buffer2[64]; // buffer for description of second device

// initialize the array of pointers BufPtrs[0] = Buffer1;

BufPtrs[1] = Buffer2;

BufPtrs[2] = NULL; // last entry should be NULL

ftStatus = FT\_ListDevices(BufPtrs,&numDevs,FT\_LIST\_ALL|FT\_OPEN\_BY\_DESCRIPTION); if (ftStatus == FT\_OK) {

// FT\_ListDevices OK, product descriptions are in Buffer1 and Buffer2, and

// numDevs contains the number of devices connected

}

else {

}

// FT\_ListDevices failed

Note that this example assumes that two devices are connected. If more devices are connected, then the size of the array of pointers must be increased and more description buffers allocated.

1. Get locations of all devices currently connected

long locIdBuf[16];

ftStatus = FT\_ListDevices(locIdBuf,&numDevs,FT\_LIST\_ALL|FT\_OPEN\_BY\_LOCATION); if (ftStatus == FT\_OK) {

// FT\_ListDevices OK, location IDs are in locIdBuf, and

// numDevs contains the number of devices connected

}

else {

}

// FT\_ListDevices failed

Note that this example assumes that no more than 16 devices are connected. If more devices are connected, then the size of the array of pointers must be increased.

## FT\_Open

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later)

Windows CE (4.2 and later)

### Summary

Open the device and return a handle which will be used for subsequent accesses.

### Definition

FT\_STATUS **FT\_Open** (int *iDevice*, FT\_HANDLE *\*ftHandle*)

### Parameters

*iDevice* Index of the device to open. Indices are 0 based.

*ftHandle* Pointer to a variable of type FT\_HANDLE where the handle will be stored. This handle must be used to access the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

Although this function can be used to open multiple devices by setting *iDevice* to 0, 1, 2 etc. there is no ability to open a specific device. To open named devices, use the function [FT\_OpenEx](#_bookmark11).

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0,&ftHandle); if (ftStatus == FT\_OK) {

// FT\_Open OK, use ftHandle to access device

}

else {

}

// FT\_Open failed

## FT\_OpenEx

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Open the specified device and return a handle that will be used for subsequent accesses. The device can

be specified by its serial number, device description or location.

This function can also be used to open multiple devices simultaneously. Multiple devices can be specified by serial number, device description or location ID (location information derived from the physical location of a device on USB). Location IDs for specific USB ports can be obtained using the utility USBView and are given in hexadecimal format. Location IDs for devices connected to a system can be obtained by calling [FT\_GetDeviceInfoList](#_bookmark7) or [FT\_ListDevices](#_bookmark9) with the appropriate flags.

### Definition

FT\_STATUS **FT\_OpenEx** (PVOID *pvArg1*, DWORD *dwFlags*, FT\_HANDLE *\*ftHandle*)

### Parameters

*pvArg1* Pointer to an argument whose type depends on the value of dwFlags. It is normally be interpreted as a pointer to a null terminated string.

*dwFlags* [FT\_OPEN\_BY\_SERIAL\_NUMBER](#_bookmark93), [FT\_OPEN\_BY\_DESCRIPTION](#_bookmark93) or [FT\_OPEN\_BY\_LOCATION](#_bookmark93).

*ftHandle* Pointer to a variable of type FT\_HANDLE where the handle will be stored. This handle must be used to access the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

The parameter specified in *pvArg1* depends on *dwFlags*: if *dwFlags* is [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93), *pvArg1* is interpreted as a pointer to a null-terminated string that represents the serial number of the device; if *dwFlags* is [*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93), *pvArg1* is interpreted as a pointer to a null- terminated string that represents the device description; if *dwFlags* is [*FT\_OPEN\_BY\_LOCATION*](#_bookmark93), *pvArg1* is interpreted as a long value that contains the location ID of the device. Please note that Windows CE and Linux do not support location IDs.

*ftHandle* is a pointer to a variable of type *FT\_HANDLE* where the handle is to be stored. This handle must be used to access the device.

### Examples

The examples that follow use these variables.

FT\_STATUS ftStatus; FT\_STATUS ftStatus2; FT\_HANDLE ftHandle1; FT\_HANDLE ftHandle2; long dwLoc;

1. Open a device with serial number "FT000001"

ftStatus = FT\_OpenEx((PVOID) "FT000001",FT\_OPEN\_BY\_SERIAL\_NUMBER,&ftHandle1); if (ftStatus == FT\_OK) {

// success - device with serial number "FT000001" is open

}

else {

}

// failure

1. Open a device with device description "USB Serial Converter"

ftStatus = FT\_OpenEx((PVOID) "USB Serial Converter",FT\_OPEN\_BY\_DESCRIPTION,&ftHandle1); if (ftStatus == FT\_OK) {

// success - device with device description "USB Serial Converter" is open

}

else {

}

// failure

1. Open 2 devices with serial numbers "FT000001" and "FT999999"

ftStatus = FT\_OpenEx((PVOID) "FT000001",FT\_OPEN\_BY\_SERIAL\_NUMBER,&ftHandle1); ftStatus2 = FT\_OpenEx((PVOID) "FT999999",FT\_OPEN\_BY\_SERIAL\_NUMBER,&ftHandle2); if (ftStatus == FT\_OK && ftStatus2 == FT\_OK) {

// success - both devices are open

}

else {

}

// failure - one or both of the devices has not been opened

1. Open 2 devices with descriptions "USB Serial Converter" and "USB Pump Controller"

ftStatus = FT\_OpenEx((PVOID) "USB Serial Converter",FT\_OPEN\_BY\_DESCRIPTION,&ftHandle1); ftStatus2 = FT\_OpenEx((PVOID) "USB Pump Controller",FT\_OPEN\_BY\_DESCRIPTION,&ftHandle2); if (ftStatus == FT\_OK && ftStatus2 == FT\_OK) {

// success - both devices are open

}

else {

}

// failure - one or both of the devices has not been opened

1. Open a device at location 23

dwLoc = 0x23;

ftStatus = FT\_OpenEx((PVOID)dwLoc,FT\_OPEN\_BY\_LOCATION,&ftHandle1); if (ftStatus == FT\_OK) {

// success - device at location 23 is open

}

else {

}

// failure

1. Open 2 devices at locations 23 and 31

dwLoc = 0x23;

ftStatus = FT\_OpenEx((PVOID) dwLoc,FT\_OPEN\_BY\_LOCATION,&ftHandle1); dwLoc = 0x31;

ftStatus2 = FT\_OpenEx((PVOID) dwLoc,FT\_OPEN\_BY\_LOCATION,&ftHandle2); if (ftStatus == FT\_OK && ftStatus2 == FT\_OK) {

// success - both devices are open

}

else {

}

// failure - one or both of the devices has not been opened

## FT\_Close

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Close an open device.

### Definition

FT\_STATUS **FT\_Close** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0,&ftHandle); if (ftStatus == FT\_OK) {

// FT\_Open OK, use ftHandle to access device

// when finished, call FT\_Close FT\_Close(ftHandle);

}

else {

}

// FT\_Open failed

## FT\_Read

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Read data from the device.

### Definition

FT\_STATUS **FT\_Read** (FT\_HANDLE *ftHandle,* LPVOID *lpBuffer,* DWORD *dwBytesToRead,*

LPDWORD lpdwBytesReturned)

### Parameters

*ftHandle* Handle of the device.

*lpBuffer* Pointer to the buffer that receives the data from the device.

*dwBytesToRead* Number of bytes to be read from the device.

*lpdwBytesReturned* Pointer to a variable of type DWORD which receives the number of

bytes read from the device.

### Return Value

FT\_OK if successful, FT\_IO\_ERROR otherwise.

### Remarks

FT\_Read always returns the number of bytes read in *lpdwBytesReturned*.

This function does not return until *dwBytesToRead* bytes have been read into the buffer. The number of bytes in the receive queue can be determined by calling [FT\_GetStatus](#_bookmark31) or [FT\_GetQueueStatus](#_bookmark26), and passed to FT\_Read as *dwBytesToRead* so that the function reads the device and returns immediately.

When a read timeout value has been specified in a previous call to [FT\_SetTimeouts](#_bookmark18), FT\_Read returns when the timer expires or *dwBytesToRead* have been read, whichever occurs first. If the timeout occurred, FT\_Read reads available data into the buffer and returns *FT\_OK*.

An application should use the function return value and *lpdwBytesReturned* when processing the buffer. If the return value is *FT\_OK*, and *lpdwBytesReturned* is equal to *dwBytesToRead* then FT\_Read has completed normally. If the return value is *FT\_OK*, and *lpdwBytesReturned* is less then *dwBytesToRead* then a timeout has occurred and the read has been partially completed. Note that if a timeout occurred and no data was read, the return value is still *FT\_OK*.

A return value of *FT\_IO\_ERROR* suggests an error in the parameters of the function, or a fatal error like a USB disconnect has occurred.

### Examples

1. This sample shows how to read all the data currently available.

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

DWORD EventDWord;

DWORD TxBytes;

DWORD RxBytes;

DWORD BytesReceived; char RxBuffer[256];

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

FT\_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord); if (RxBytes > 0) {

ftStatus = FT\_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived); if (ftStatus == FT\_OK) {

// FT\_Read OK

}

else {

}

// FT\_Read Failed

}

FT\_Close(ftHandle);

1. This sample shows how to read with a timeout of 5 seconds.

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; DWORD RxBytes = 10; DWORD BytesReceived; char RxBuffer[256];

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

FT\_SetTimeouts(ftHandle,5000,0);

ftStatus = FT\_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived); if (ftStatus == FT\_OK) {

if (BytesReceived == RxBytes) {

// FT\_Read OK

}

else {

}

}

// FT\_Read Timeout

else {

}

// FT\_Read Failed

FT\_Close(ftHandle);

## FT\_Write

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Write data to the device.

### Definition

FT\_STATUS **FT\_Write** (FT\_HANDLE *ftHandle,* LPVOID *lpBuffer,* DWORD *dwBytesToWrite,*

LPDWORD lpdwBytesWritten)

### Parameters

*ftHandle* Handle of the device.

*lpBuffer* Pointer to the buffer that contains the data to be written to the device.

*dwBytesToWrite* Number of bytes to write to the device.

*lpdwBytesWritten* Pointer to a variable of type DWORD which receives the number of

bytes written to the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; DWORD BytesWritten;

char TxBuffer[256]; // Contains data to write to device

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_Write(ftHandle, TxBuffer, sizeof(TxBuffer), &BytesWritten); if (ftStatus == FT\_OK) {

// FT\_Write OK

}

else {

}

// FT\_Write Failed

FT\_Close(ftHandle);

## FT\_SetBaudRate

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the baud rate for the device.

### Definition

FT\_STATUS **FT\_SetBaudRate** (FT\_HANDLE *ftHandle,* DWORD *dwBaudRate*)

### Parameters

*ftHandle* Handle of the device.

*dwBaudRate* Baud rate.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetBaudRate(ftHandle, 115200); // Set baud rate to 115200 if (ftStatus == FT\_OK) {

// FT\_SetBaudRate OK

}

else {

}

// FT\_SetBaudRate Failed

FT\_Close(ftHandle);

## FT\_SetDivisor

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the baud rate for the device. It is used to set non-standard baud rates.

### Definition

FT\_STATUS **FT\_SetDivisor** (FT\_HANDLE *ftHandle,* USHORT *usDivisor*)

### Parameters

*ftHandle* Handle of the device.

*usDivisor* Divisor.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is no longer required as [FT\_SetBaudRate](#_bookmark15) will now automatically calculate the required divisor for a requested baud rate. The application note "Setting baud rates for the FT8U232AM" is available from the Application Notes section of the FTDI website describes how to calculate the divisor for a non-standard baud rate.

## FT\_SetDataCharacteristics

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

This function sets the data characteristics for the device.

### Definition

FT\_STATUS **FT\_SetDataCharacteristics** (FT\_HANDLE *ftHandle,* UCHAR *uWordLength,*

UCHAR uStopBits, UCHAR uParity)

### Parameters

*ftHandle* Handle of the device.

*uWordLength* Number of bits per word - must be [FT\_BITS\_8](#_bookmark94) or [FT\_BITS\_7](#_bookmark94).

*uStopBits* Number of stop bits - must be [FT\_STOP\_BITS\_1](#_bookmark95) or [FT\_STOP\_BITS\_2](#_bookmark95).

*uParity* Parity - must be [FT\_PARITY\_NONE](#_bookmark96), [FT\_PARITY\_ODD](#_bookmark96), [FT\_PARITY\_EVEN](#_bookmark96), [FT\_PARITY\_MARK](#_bookmark96) or [FT\_PARITY SPACE](#_bookmark96).

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

// Set 8 data bits, 1 stop bit and no parity

ftStatus = FT\_SetDataCharacteristics(ftHandle, FT\_BITS\_8, FT\_STOP\_BITS\_1, FT\_PARITY\_NONE);

if (ftStatus == FT\_OK) {

// FT\_SetDataCharacteristics OK

}

else {

}

// FT\_SetDataCharacteristics Failed

FT\_Close(ftHandle);

## FT\_SetTimeouts

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the read and write timeouts for the device.

### Definition

FT\_STATUS **FT\_SetTimeouts** (FT\_HANDLE *ftHandle,* DWORD *dwReadTimeout,*

DWORD dwWriteTimeout)

### Parameters

*ftHandle* Handle of the device.

*dwReadTimeout* Read timeout in milliseconds.

*dwWriteTimeout* Write timeout in milliseconds.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

// Set read timeout of 5sec, write timeout of 1sec ftStatus = FT\_SetTimeouts(ftHandle, 5000, 1000); if (ftStatus == FT\_OK) {

// FT\_SetTimeouts OK

}

else {

}

// FT\_SetTimeouts failed

FT\_Close(ftHandle);

## FT\_SetFlowControl

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the flow control for the device.

### Definition

FT\_STATUS **FT\_SetFlowControl** (FT\_HANDLE *ftHandle,* USHORT *usFlowControl,*

UCHAR *uXon,* UCHAR *uXoff*)

### Parameters

*ftHandle* Handle of the device.

*usFlowControl* Must be one of [FT\_FLOW\_NONE](#_bookmark97), [FT\_FLOW\_RTS\_CTS](#_bookmark97), [FT\_FLOW\_DTR\_DSR](#_bookmark97) or [FT\_FLOW\_XON\_XOFF](#_bookmark97).

*uXon* Character used to signal Xon. Only used if flow control is [FT\_FLOW\_XON\_XOFF](#_bookmark97).

*uXoff* Character used to signal Xoff. Only used if flow control is [FT\_FLOW\_XON\_XOFF](#_bookmark97).

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

// Set RTS/CTS flow control

ftStatus = FT\_SetFlowControl(ftHandle, FT\_FLOW\_RTS\_CTS, 0x11, 0x13); if (ftStatus == FT\_OK) {

// FT\_SetFlowControl OK

}

else {

}

// FT\_SetFlowControl Failed

FT\_Close(ftHandle);

## FT\_SetDtr

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the Data Terminal Ready (DTR) control signal.

### Definition

FT\_STATUS **FT\_SetDtr** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function asserts the Data Terminal Ready (DTR) line of the device.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetDtr(ftHandle); if (ftStatus == FT\_OK) {

// FT\_SetDtr OK

}

else {

}

// FT\_SetDtr failed

FT\_Close(ftHandle);

## FT\_ClrDtr

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

This function clears the Data Terminal Ready (DTR) control signal.

### Definition

FT\_STATUS **FT\_ClrDtr** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function de-asserts the Data Terminal Ready (DTR) line of the device.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_ClrDtr(ftHandle); if (ftStatus == FT\_OK) {

// FT\_ClrDtr OK

}

else {

}

// FT\_ClrDtr failed

FT\_Close(ftHandle);

## FT\_SetRts

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the Request To Send (RTS) control signal.

### Definition

FT\_STATUS **FT\_SetRts** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function asserts the Request To Send (RTS) line of the device.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetRts(ftHandle); if (ftStatus == FT\_OK) {

// FT\_SetRts OK

}

else {

}

// FT\_SetRts failed

FT\_Close(ftHandle);

## FT\_ClrRts

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function clears the Request To Send (RTS) control signal.

### Definition

FT\_STATUS **FT\_ClrRts** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function de-asserts the Request To Send (RTS) line of the device.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_ClrRts(ftHandle); if (ftStatus == FT\_OK) {

// FT\_ClrRts OK

}

else {

}

// FT\_ClrRts failed

FT\_Close(ftHandle);

## FT\_GetModemStatus

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

Gets the modem status and line status from the device.

### Definition

FT\_STATUS **FT\_GetModemStatus** (FT\_HANDLE *ftHandle,* LPDWORD *lpdwModemStatus*)

### Parameters

*ftHandle* Handle of the device.

*lpdwModemStatus* Pointer to a variable of type DWORD which receives the modem

status and line status from the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

The least significant byte of the *lpdwModemStatus* value holds the modem status. On Windows and Windows CE, the line status is held in the second least significant byte of the *lpdwModemStatus* value.

The modem status is bit-mapped as follows: Clear To Send ([CTS](#_bookmark100)) = 0x10, Data Set Ready ([DSR](#_bookmark100)) = 0x20, Ring Indicator ([RI](#_bookmark100)) = 0x40, Data Carrier Detect ([DCD](#_bookmark100)) = 0x80.

The line status is bit-mapped as follows: Overrun Error ([OE](#_bookmark101)) = 0x02, Parity Error ([PE](#_bookmark101)) = 0x04, Framing Error ([FE](#_bookmark101)) = 0x08, Break Interrupt ([BI](#_bookmark101)) = 0x10.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; DWORD dwModemStatus = 0; DWORD dwLineStatus = 0;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_GetModemStatus(ftHandle, &dwModemStatus); if (ftStatus == FT\_OK) {

// FT\_GetModemStatus OK

// Line status is the second byte of the dwModemStatus value dwLineStatus = ((dwModemStatus >> 8) & 0x000000FF);

// Now mask off the modem status byte dwModemStatus = (dwModemStatus & 0x000000FF);

}

else {

}

// FT\_GetModemStatus failed

FT\_Close(ftHandle);

## FT\_GetQueueStatus

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

Gets the number of bytes in the receive queue.

### Definition

FT\_STATUS **FT\_GetQueueStatus** (FT\_HANDLE *ftHandle,* LPDWORD *lpdwAmountInRxQueue*)

### Parameters

*ftHandle* Handle of the device.

*lpdwAmountInRxQueue* Pointer to a variable of type DWORD which receives the number of

bytes in the receive queue.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; DWORD RxBytes;

DWORD BytesReceived; char RxBuffer[256];

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

FT\_GetQueueStatus(ftHandle,&RxBytes); if (RxBytes > 0) {

ftStatus = FT\_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived); if (ftStatus == FT\_OK) {

// FT\_Read OK

}

else {

}

}

// FT\_Read Failed

FT\_Close(ftHandle);

## FT\_GetDeviceInfo

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Get device information for an open device.

### Definition

FT\_STATUS **FT\_GetDeviceInfo** (FT\_HANDLE *ftHandle,* FT\_DEVICE *\*pftType,*

LPDWORD *lpdwID*, PCHAR pcSerialNumber, PCHAR pcDescription, PVOID pvDummy)

### Parameters

*ftHandle* Handle of the device.

*pftType* Pointer to unsigned long to store device type.

*lpdwID* Pointer to unsigned long to store device ID.

*pcSerialNumber* Pointer to buffer to store device serial number as a null-

terminated string.

*pcDescription* Pointer to buffer to store device description as a null-terminated string.

*pvDummy* Reserved for future use - should be set to NULL.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is used to return the device type, device ID, device description and serial number.

The device ID is encoded in a DWORD - the most significant word contains the vendor ID, and the least significant word contains the product ID. So the returned ID 0x04036001 corresponds to the device ID VID\_0403&PID\_6001.

### Example

FT\_HANDLE ftHandle; FT\_DEVICE ftDevice; FT\_STATUS ftStatus; DWORD deviceID;

char SerialNumber[16]; char Description[64];

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_GetDeviceInfo(

ftHandle, &ftDevice, &deviceID, SerialNumber, Description, NULL

);

if (ftStatus == FT\_OK) {

if (ftDevice == FT\_DEVICE\_232H)

; // device is FT232H

else if (ftDevice == FT\_DEVICE\_4232H)

; // device is FT4232H

else if (ftDevice == FT\_DEVICE\_2232H)

; // device is FT2232H

else if (ftDevice == FT\_DEVICE\_232R)

; // device is FT232R

else if (ftDevice == FT\_DEVICE\_2232C)

; // device is FT2232C/L/D else if (ftDevice == FT\_DEVICE\_BM)

; // device is FTU232BM else if (ftDevice == FT\_DEVICE\_AM)

; // device is FT8U232AM

else

; // unknown device (this should not happen!)

}

else {

}

// deviceID contains encoded device ID

// SerialNumber, Description contain 0-terminated strings

// FT\_GetDeviceType FAILED!

FT\_Close(ftHandle);

## FT\_GetDriverVersion

### Supported Operating Systems

Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function returns the D2XX driver version number.

### Definition

FT\_STATUS **FT\_GetDriverVersion** (FT\_HANDLE *ftHandle*, LPDWORD *lpdwDriverVersion*)

### Parameters

*ftHandle* Handle of the device. *lpdwDriverVersion* Pointer to the driver version number. **Return Value**

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

A version number consists of major, minor and build version numbers contained in a 4-byte field (unsigned long). Byte0 (least significant) holds the build version, Byte1 holds the minor version, and Byte2 holds the major version. Byte3 is currently set to zero.

For example, driver version "2.04.06" is represented as 0x00020406. Note that a device has to be opened before this function can be called.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; DWORD dwDriverVer;

// Get driver version

ftStatus = FT\_Open(0,&ftHandle); if (ftStatus == FT\_OK) {

ftStatus = FT\_GetDriverVersion(ftHandle,&dwDriverVer); if (ftStatus == FT\_OK)

printf("Driver version = 0x%x\n",dwDriverVer);

else

printf("error reading driver version\n");

FT\_Close(ftHandle);

}

## FT\_GetLibraryVersion

### Supported Operating Systems

Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function returns D2XX DLL version number.

### Definition

FT\_STATUS **FT\_GetLibraryVersion** (LPDWORD *lpdwDLLVersion*)

### Parameters

*lpdwDLLVersion* Pointer to the DLL version number.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

A version number consists of major, minor and build version numbers contained in a 4-byte field (unsigned long). Byte0 (least significant) holds the build version, Byte1 holds the minor version, and Byte2 holds the major version. Byte3 is currently set to zero.

For example, D2XX DLL version "3.01.15" is represented as 0x00030115. Note that this function does not take a handle, and so it can be called without opening a device.

### Example

FT\_STATUS ftStatus;

DWORD dwLibraryVer;

// Get DLL version

ftStatus = FT\_GetLibraryVersion(&dwLibraryVer); if (ftStatus == FT\_OK)

printf("Library version = 0x%x\n",dwLibraryVer);

else

printf("error reading library version\n");

## FT\_GetComPortNumber

**Supported Operating Systems** Windows (2000 and later) **Summary**

Retrieves the COM port associated with a device.

### Definition

FT\_STATUS **FT\_GetComPortNumber** (FT\_HANDLE *ftHandle,* LPLONG *lplComPortNumber*)

### Parameters

*ftHandle* Handle of the device.

*lplComPortNumber* Pointer to a variable of type LONG which receives the COM port number associated with the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is only available when using the Windows CDM driver as both the D2XX and VCP drivers can be installed at the same time.

If no COM port is associated with the device, *lplComPortNumber* will have a value of -1.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); LONG lComPortNumber;

ftStatus = FT\_GetComPortNumber(ftHandle,&lComPortNumber); if (ftStatus == FT\_OK) {

if (lComPortNumber == -1) {

// No COM port assigned

}

else {

}

else {

}

// COM port assigned with number held in lComPortNumber

// FT\_GetComPortNumber FAILED!

}

FT\_Close(ftHandle);

## FT\_GetStatus

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Gets the device status including number of characters in the receive queue, number of characters in the

transmit queue, and the current event status.

### Definition

FT\_STATUS **FT\_GetStatus** (FT\_HANDLE *ftHandle,* LPDWORD *lpdwAmountInRxQueue*,

LPDWORD lpdwAmountInTxQueue, LPDWORD lpdwEventStatus)

### Parameters

*ftHandle* Handle of the device.

*lpdwAmountInRxQueue* Pointer to a variable of type DWORD which receives the number of characters in the receive queue.

*lpdwAmountInTxQueue* Pointer to a variable of type DWORD which receives the number of characters in the transmit queue.

*lpdwEventStatus* Pointer to a variable of type DWORD which receives the current state of the event status.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

For an example of how to use this function, see the sample code in [FT\_SetEventNotification](#_bookmark32).

## FT\_SetEventNotification

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Sets conditions for event notification.

### Definition

FT\_STATUS **FT\_SetEventNotification** (FT\_HANDLE *ftHandle,* DWORD *dwEventMask,*

PVOID *pvArg*)

### Parameters

*ftHandle* Handle of the device.

*dwEventMask* Conditions that cause the event to be set.

*pvArg* Interpreted as the handle of an event.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

An application can use 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, then block on the event. When the conditions are met, the event is set, and the application thread unblocked.

*dwEventMask* is a bit-map that describes the events the application is interested in. *pvArg* is interpreted as the handle of an event which has been created by the application. If one of the event conditions is met, the event is set.

If [*FT\_EVENT\_RXCHAR*](#_bookmark99) is set in *dwEventMask*, the event will be set when a character has been received by the device.

If [*FT\_EVENT\_MODEM\_STATUS*](#_bookmark99) is set in *dwEventMask*, the event will be set when a change in the modem signals has been detected by the device.

If [*FT\_EVENT\_LINE\_STATUS*](#_bookmark99) is set in *dwEventMask*, the event will be set when a change in the line status has been detected by the device.

### Examples

1. This example is valid for Windows and Windows CE and shows how to wait for a character to be received or a change in modem status.

// First, create the event and call FT\_SetEventNotification. FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); HANDLE hEvent;

DWORD EventMask;

hEvent = CreateEvent(

NULL,

false, // auto-reset event false, // non-signalled state (LPCWSTR) ""

);

EventMask = FT\_EVENT\_RXCHAR | FT\_EVENT\_MODEM\_STATUS;

ftStatus = FT\_SetEventNotification(ftHandle,EventMask,hEvent);

// Sometime later, block the application thread by waiting on the event, then when the event has

// occurred, determine the condition that caused the event, and process it accordingly. WaitForSingleObject(hEvent,INFINITE);

DWORD EventDWord;

DWORD RxBytes;

DWORD TxBytes;

FT\_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord); if (EventDWord & FT\_EVENT\_MODEM\_STATUS) {

// modem status event detected, so get current modem status FT\_GetModemStatus(ftHandle,&Status);

if (Status & 0x00000010) {

// CTS is high

}

else {

}

// CTS is low

if (Status & 0x00000020) {

// DSR is high

}

else {

}

}

// DSR is low

if (RxBytes > 0) {

// call FT\_Read() to get received data from device

}

1. This example is valid for Linux and shows how to wait for a character to be received or a change in modem status.

// First, create the event and call FT\_SetEventNotification. FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

EVENT\_HANDLE eh;

DWORD EventMask;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

pthread\_mutex\_init(&eh.eMutex, NULL); pthread\_cond\_init(&eh.eCondVar, NULL);

EventMask = FT\_EVENT\_RXCHAR | FT\_EVENT\_MODEM\_STATUS;

ftStatus = FT\_SetEventNotification(ftHandle, EventMask, (PVOID)&eh);

// Sometime later, block the application thread by waiting on the event, then when the event has

// occurred, determine the condition that caused the event, and process it accordingly.

pthread\_mutex\_lock(&eh.eMutex); pthread\_cond\_wait(&eh.eCondVar, &eh.eMutex); pthread\_mutex\_unlock(&eh.eMutex);

DWORD EventDWord;

DWORD RxBytes;

DWORD TxBytes;

DWORD Status;

FT\_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord); if (EventDWord & FT\_EVENT\_MODEM\_STATUS) {

// modem status event detected, so get current modem status FT\_GetModemStatus(ftHandle,&Status);

if (Status & 0x00000010) {

// CTS is high

}

else {

}

// CTS is low

if (Status & 0x00000020) {

// DSR is high

}

else {

}

}

// DSR is low

if (RxBytes > 0) {

// call FT\_Read() to get received data from device

}

FT\_Close(ftHandle);

## FT\_SetChars

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

This function sets the special characters for the device.

### Definition

FT\_STATUS **FT\_SetChars** (FT\_HANDLE *ftHandle,* UCHAR *uEventCh,* UCHAR *uEventChEn,*

UCHAR uErrorCh, UCHAR uErrorChEn)

### Parameters

*ftHandle* Handle of the device.

*uEventCh* Event character.

*uEventChEn* 0 if event character disabled, non-zero otherwise.

*uErrorCh* Error character.

*uErrorChEn* 0 if error character disabled, non-zero otherwise.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function allows for inserting specified characters in the data stream to represent events firing or errors occurring.

## FT\_SetBreakOn

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Sets the BREAK condition for the device.

### Definition

FT\_STATUS **FT\_SetBreakOn** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); ftStatus = FT\_SetBreakOn(ftHandle);

if (ftStatus == FT\_OK) {

// FT\_SetBreakOn OK

}

else {

}

// FT\_SetBreakOn failed

FT\_Close(ftHandle);

## FT\_SetBreakOff

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Resets the BREAK condition for the device.

### Definition

FT\_STATUS **FT\_SetBreakOff** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); ftStatus = FT\_SetBreakOff(ftHandle);

if (ftStatus == FT\_OK) {

// FT\_SetBreakOff OK

}

else {

}

// FT\_SetBreakOff failed

FT\_Close(ftHandle);

## FT\_Purge

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function purges receive and transmit buffers in the device.

### Definition

FT\_STATUS **FT\_Purge** (FT\_HANDLE *ftHandle,* DWORD *dwMask*)

### Parameters

*ftHandle* Handle of the device.

*uEventCh* Combination of [FT\_PURGE\_RX](#_bookmark98) and [FT\_PURGE\_TX](#_bookmark98).

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

ftStatus = FT\_Purge(ftHandle, FT\_PURGE\_RX | FT\_PURGE\_TX); // Purge both Rx and Tx buffers if (ftStatus == FT\_OK) {

// FT\_Purge OK

}

else {

}

// FT\_Purge failed

FT\_Close(ftHandle);

## FT\_ResetDevice

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sends a reset command to the device.

### Definition

FT\_STATUS **FT\_ResetDevice** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); ftStatus = FT\_ResetDevice(ftHandle);

if (ftStatus == FT\_OK) {

// FT\_ResetDevice OK

}

else {

}

// FT\_ResetDevice failed

FT\_Close(ftHandle);

## FT\_ResetPort

**Supported Operating Systems** Windows (2000 and later) **Summary**

Send a reset command to the port.

### Definition

FT\_STATUS **FT\_ResetPort** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is used to attempt to recover the port after a failure. It is not equivalent to an unplug- replug event. For the equivalent of an unplug-replug event, use [FT\_CyclePort](#_bookmark39).

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

ftStatus = FT\_ResetPort(ftHandle); if (ftStatus == FT\_OK) {

// Port has been reset

}

else {

}

// FT\_ResetPort FAILED!

## FT\_CyclePort

**Supported Operating Systems** Windows (2000 and later) **Summary**

Send a cycle command to the USB port.

### Definition

FT\_STATUS **FT\_CyclePort** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

The effect of this function is the same as disconnecting then reconnecting the device from USB. Possible use of this function is situations where a fatal error has occurred and it is difficult, or not possible, to recover without unplugging and replugging the USB cable. This function can also be used after re- programming the EEPROM to force the FTDI device to read the new EEPROM contents which would otherwise require a physical disconnect-reconnect.

As the current session is not restored when the driver is reloaded, the application must be able to recover after calling this function. It is ithe responisbility of the application to close the handle after successfully calling FT\_CyclePort.

For FT4232H, FT2232H and FT2232 devices, FT\_CyclePort will only work under Windows XP and later.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

ftStatus = FT\_CyclePort(ftHandle); if (ftStatus == FT\_OK) {

// Port has been cycled.

// Close the handle.

ftStatus = FT\_Close(ftHandle);

}

else {

}

// FT\_CyclePort FAILED!

## FT\_Rescan

**Supported Operating Systems** Windows (2000 and later) **Summary**

This function can be of use when trying to recover devices programatically.

### Definition

FT\_STATUS **FT\_Rescan** ()

### Parameters

None

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

Calling FT\_Rescan is equivalent to clicking the "Scan for hardware changes" button in the Device Manager. Only USB hardware is checked for new devices. All USB devices are scanned, not just FTDI devices.

### Example

FT\_STATUS ftStatus;

ftStatus = FT\_Rescan(); if(ftStatus != FT\_OK) {

// FT\_Rescan failed! return;

}

## FT\_Reload

**Supported Operating Systems** Windows (2000 and later) **Summary**

This function forces a reload of the driver for devices with a specific VID and PID combination.

### Definition

FT\_STATUS **FT\_Reload** (WORD *wVID*, WORD *wPID*)

### Parameters

*wVID* Vendor ID of the devices to reload the driver for.

*wPID* Product ID of the devices to reload the driver for.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

Calling FT\_Reload forces the operating system to unload and reload the driver for the specified device IDs. If the VID and PID parameters are null, the drivers for USB root hubs will be reloaded, causing all USB devices connected to reload their drivers. Please note that this function will not work correctly on 64-bit Windows when called from a 32-bit application.

### Examples

1. This example shows how to call FT\_Reload to reload the driver for a standard FT232R device (VID 0x0403, PID 0x6001).

FT\_STATUS ftStatus; WORD wVID = 0x0403; WORD wPID = 0x6001;

ftStatus = FT\_Reload(wVID,wPID); if(ftStatus != FT\_OK) {

// FT\_Reload failed! return;

}

1. This example shows how to call FT\_Reload to reload the drivers for all USB devices. FT\_STATUS ftStatus;

WORD wVID = 0x0000; WORD wPID = 0x0000;

ftStatus = FT\_Reload(wVID,wPID); if(ftStatus != FT\_OK) {

// FT\_Reload failed! return;

}

## FT\_SetResetPipeRetryCount

### Supported Operating Systems

Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Set the ResetPipeRetryCount value.

### Definition

FT\_STATUS **FT\_SetResetPipeRetryCount** (FT\_HANDLE *ftHandle,* DWORD *dwCount*)

### Parameters

*ftHandle* Handle of the device.

*dwCount* Unsigned long containing required ResetPipeRetryCount.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is used to set the ResetPipeRetryCount. ResetPipeRetryCount controls the maximum number of times that the driver tries to reset a pipe on which an error has occurred.

ResetPipeRequestRetryCount defaults to 50. It may be necessary to increase this value in noisy environments where a lot of USB errors occur.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); DWORD dwRetryCount;

dwRetryCount = 100;

ftStatus = FT\_SetResetPipeRetryCount(ftHandle,dwRetryCount); if (ftStatus == FT\_OK) {

// ResetPipeRetryCount set to 100

}

else {

}

// FT\_SetResetPipeRetryCount FAILED!

## FT\_StopInTask

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Stops the driver's IN task.

### Definition

FT\_STATUS **FT\_StopInTask** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is 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 [FT\_RestartInTask](#_bookmark44) which sets the IN task running again.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

do {

ftStatus = FT\_StopInTask(ftHandle);

} while (ftStatus != FT\_OK);

//

// Do something - for example purge device

//

do {

ftStatus = FT\_RestartInTask(ftHandle);

} while (ftStatus != FT\_OK); FT\_Close(ftHandle);

## FT\_RestartInTask

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Restart the driver's IN task.

### Definition

FT\_STATUS **FT\_RestartInTask** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function is used to restart the driver's IN task (read) after it has been stopped by a call to [FT\_StopInTask](#_bookmark43).

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

do {

ftStatus = FT\_StopInTask(ftHandle);

} while (ftStatus != FT\_OK);

//

// Do something - for example purge device

//

do {

ftStatus = FT\_RestartInTask(ftHandle);

} while (ftStatus != FT\_OK); FT\_Close(ftHandle);

## FT\_SetDeadmanTimeout

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function allows the maximum time in milliseconds that a USB request can remain outstanding to be

set.

### Definition

FT\_STATUS **FT\_SetDeadmanTimeout** (FT\_HANDLE *ftHandle*, DWORD *dwDeadmanTimeout*)

### Parameters

*ftHandle* Handle of the device.

*dwDeadmanTimeout* Deadman timeout value in milliseconds. Default value is 5000.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

The deadman timeout is referred to in application note AN232B-10 Advanced Driver Options from the FTDI web site as the USB timeout. It is unlikely that this function will be required by most users.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

DWORD dwDeadmanTimeout = 6000;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetDeadmanTimeout(ftHandle,dwDeadmanTimeout); if (ftStatus == FT\_OK) {

// Set Deadman Timer to 6 seconds

}

else {

}

// FT\_SetDeadmanTimeout FAILED!

FT\_Close(ftHandle);

## FT\_IoCtl

Undocumented function.

## FT\_SetWaitMask

Undocumented function.

## FT\_WaitOnMask

Undocumented function.

# EEPROM Programming Interface Functions

FTDI device EEPROMs can be both read and programmed using the functions listed in this section. Please note the following information:

* The Maximum length of the Manufacturer, ManufacturerId, Description and SerialNumber strings is 48 words (1 word = 2 bytes).
* The first two characters of the serial number are the manufacturer ID.
* The Manufacturer string length plus the Description string length is less than or equal to 40 characters with the following functions:
  + [FT\_EE\_Read](#_bookmark53)
  + [FT\_EE\_Program](#_bookmark55)
  + [FT\_EE\_ProgramEx](#_bookmark56)
  + [FT\_EEPROM\_Read](#_bookmark60)
  + [FT\_EEPROM\_Program](#_bookmark61)
* The serial number should be maximum 15 characters long on single port devices (eg FT232R, FT-

X) and 14 characters on multi port devices (eg FT2232H, FT4232H). If it is longer then it may be truncated and will not have a null terminator.

For instance a serial number which is 15 characters long on a multi-port device will have an effective serial number which is 16 characters long since the serial number is appended with the channel identifier (A,B,etc). The buffer used to return the string from the API is only 16 characters in size so the NULL termination will be lost.

If the serial number or description are too long in the EEPROM or configuration of a device then the strings returned by FT\_GetDeviceInfo and FT\_ListDevices may not be NULL terminated

## FT\_ReadEE

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Read a value from an EEPROM location.

### Definition

FT\_STATUS **FT\_ReadEE** (FT\_HANDLE *ftHandle*, DWORD *dwWordOffset*, LPWORD *lpwValue*)

### Parameters

*ftHandle* Handle of the device.

*dwWordOffset* EEPROM location to read from.

*lpwValue* Pointer to the WORD value read from the EEPROM.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

EEPROMs for FTDI devices are organised by WORD, so each value returned is 16-bits wide.

## FT\_WriteEE

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Write a value to an EEPROM location.

### Definition

FT\_STATUS **FT\_WriteEE** (FT\_HANDLE *ftHandle*, DWORD *dwWordOffset*, WORD *wValue*)

### Parameters

*ftHandle* Handle of the device.

*dwWordOffset* EEPROM location to read from.

*wValue* The WORD value write to the EEPROM.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

EEPROMs for FTDI devices are organised by WORD, so each value written to the EEPROM is 16-bits wide.

## FT\_EraseEE

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Erases the device EEPROM.

### Definition

FT\_STATUS **FT\_EraseEE** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function will erase the entire contents of an EEPROM, including the user area. Note that the FT232R and FT245R devices have an internal EEPROM that cannot be erased.

## FT\_EE\_Read

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Read the contents of the EEPROM.

### Definition

FT\_STATUS **FT\_EE\_Read** (FT\_HANDLE *ftHandle*, PFT\_PROGRAM\_DATA *pData*)

### Parameters

*ftHandle* Handle of the device.

*pData* Pointer to structure of type FT\_PROGRAM\_DATA.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function interprets the parameter *pData* as a pointer to a structure of type *FT\_PROGRAM\_DATA* that contains storage for the data to be read from the EEPROM.

The function does not perform any checks on buffer sizes, so the buffers passed in the *FT\_PROGRAM\_DATA* structure must be big enough to accommodate their respective strings (including null terminators). The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the *Manufacturer* string length plus the *Description* string length is less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT\_PROGRAM\_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT\_PROGRAM\_DATA* structure which is defined in FTD2XX.h.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); if (ftStatus != FT\_OK) {

// FT\_Open FAILED!

}

FT\_PROGRAM\_DATA ftData; char ManufacturerBuf[32]; char ManufacturerIdBuf[16]; char DescriptionBuf[64]; char SerialNumberBuf[16];

ftData.Signature1 = 0x00000000; ftData.Signature2 = 0xffffffff;

ftData.Version = 0x00000005; // EEPROM structure with FT232H extensions ftData.Manufacturer = ManufacturerBuf;

ftData.ManufacturerId = ManufacturerIdBuf; ftData.Description = DescriptionBuf; ftData.SerialNumber = SerialNumberBuf;

ftStatus = FT\_EE\_Read(ftHandle, &ftData); if (ftStatus == FT\_OK) {

// FT\_EE\_Read OK, data is available in ftData

}

else {

}

// FT\_EE\_Read FAILED!

FT\_Close(ftHandle);

## FT\_EE\_ReadEx

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Read the contents of the EEPROM and pass strings separately.

### Definition

FT\_STATUS **FT\_EE\_ReadEx** (FT\_HANDLE *ftHandle*, PFT\_PROGRAM\_DATA *pData*,

char *\*Manufacturer*, char \*ManufacturerId, char \*Description, char \*SerialNumber)

### Parameters

*ftHandle* Handle of the device.

*pData* Pointer to structure of type FT\_PROGRAM\_DATA.

*\*Manufacturer* Pointer to a null-terminated string containing the manufacturer

name.

*\*ManufacturerId* Pointer to a null-terminated string containing the manufacturer ID.

*\*Description* Pointer to a null-terminated string containing the device description.

*\*SerialNumber* Pointer to a null-terminated string containing the device serial

number.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This variation of the standard [FT\_EE\_Read](#_bookmark53) function was included to provide support for languages such as LabVIEW where problems can occur when string pointers are contained in a structure.

This function interprets the parameter *pData* as a pointer to a structure of type *FT\_PROGRAM\_DATA* that contains storage for the data to be read from the EEPROM.

The function does not perform any checks on buffer sizes, so the buffers passed in the *FT\_PROGRAM\_DATA* structure must be big enough to accommodate their respective strings (including null terminators).

Note that the DLL must be informed which version of the *FT\_PROGRAM\_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT\_PROGRAM\_DATA* structure which is defined in FTD2XX.h.

The string parameters in the *FT\_PROGRAM\_DATA* structure should be passed as DWORDs to avoid overlapping of parameters. All string pointers are passed out separately from the *FT\_PROGRAM\_DATA* structure.

## FT\_EE\_Program

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Program the EEPROM.

### Definition

FT\_STATUS **FT\_EE\_Program** (FT\_HANDLE *ftHandle*, PFT\_PROGRAM\_DATA *pData*)

### Parameters

*ftHandle* Handle of the device.

*pData* Pointer to structure of type FT\_PROGRAM\_DATA.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function interprets the parameter *pData* as a pointer to a structure of type *FT\_PROGRAM\_DATA* that contains the data to write to the EEPROM. The data is written to EEPROM, then read back and verified.

If the *SerialNumber* field in *FT\_PROGRAM\_DATA* is NULL, or *SerialNumber* points to a NULL string, a serial number based on the *ManufacturerId* and the current date and time will be generated. The *Manufacturer* string length plus the *Description* string length must be less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT\_PROGRAM\_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT\_PROGRAM\_DATA* structure which is defined in FTD2XX.h.

If *pData* is NULL, the structure version will default to 0 (original BM series) and the device will be programmed with the default data:

### Example

This example shows how to program the EEPROM of an FT232B device. Other parameters would need to be set up for other device types.

// Version 4 structure for programming a BM device.

// Other elements would need non-zero values for FT2232, FT232R, FT245R, FT2232H or

// FT4232H devices.

FT\_PROGRAM\_DATA ftData = {

0x00000000, // Header - must be 0x00000000

0xFFFFFFFF, // Header - must be 0xffffffff

0x00000004, // Header - FT\_PROGRAM\_DATA version 0x0403, // VID

0x6001, // PID

(char \*) "FTDI", // Manufacturer

(char \*) "FT", // Manufacturer ID (char \*) "USB HS Serial Converter", // Description (char \*) "FT000001", // Serial Number 44, // MaxPower

1, // PnP

0, // SelfPowered

1, // RemoteWakeup

1, // non-zero if Rev4 chip, zero otherwise

0, // non-zero if in endpoint is isochronous

0, // non-zero if out endpoint is isochronous

0, // non-zero if pull down enabled

1, // non-zero if serial number to be used

0, // non-zero if chip uses USBVersion

0x0110, // BCD (0x0200 => USB2)

//

// FT2232C extensions (Enabled if Version = 1 or greater)

//

0, // non-zero if Rev5 chip, zero otherwise

0, // non-zero if in endpoint is isochronous

0, // non-zero if in endpoint is isochronous

0, // non-zero if out endpoint is isochronous

0, // non-zero if out endpoint is isochronous

0, // non-zero if pull down enabled

0, // non-zero if serial number to be used

0, // non-zero if chip uses USBVersion

0x0, // BCD (0x0200 => USB2)

0, // non-zero if interface is high current

0, // non-zero if interface is high current

0, // non-zero if interface is 245 FIFO

0, // non-zero if interface is 245 FIFO CPU target

0, // non-zero if interface is Fast serial

0, // non-zero if interface is to use VCP drivers

0, // non-zero if interface is 245 FIFO

0, // non-zero if interface is 245 FIFO CPU target

0, // non-zero if interface is Fast serial

0, // non-zero if interface is to use VCP drivers

//

// FT232R extensions (Enabled if Version = 2 or greater)

//

0, // Use External Oscillator

0, // High Drive I/Os

0, // Endpoint size

0, // non-zero if pull down enabled

0, // non-zero if serial number to be used

0, // non-zero if invert TXD

0, // non-zero if invert RXD

0, // non-zero if invert RTS

0, // non-zero if invert CTS

0, // non-zero if invert DTR

0, // non-zero if invert DSR

0, // non-zero if invert DCD

0, // non-zero if invert RI

0, // Cbus Mux control

0, // Cbus Mux control

0, // Cbus Mux control

0, // Cbus Mux control

0, // Cbus Mux control

0, // non-zero if using D2XX drivers

//

// Rev 7 (FT2232H) Extensions (Enabled if Version = 3 or greater)

//

0, // non-zero if pull down enabled

0, // non-zero if serial number to be used

0, // non-zero if AL pins have slow slew

0, // non-zero if AL pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if AH pins have slow slew

0, // non-zero if AH pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if BL pins have slow slew

0, // non-zero if BL pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if BH pins have slow slew

0, // non-zero if BH pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if interface is 245 FIFO

0, // non-zero if interface is 245 FIFO CPU target

0, // non-zero if interface is Fast serial

0, // non-zero if interface is to use VCP drivers

0, // non-zero if interface is 245 FIFO

0, // non-zero if interface is 245 FIFO CPU target

0, // non-zero if interface is Fast serial

0, // non-zero if interface is to use VCP drivers

0, // non-zero if using BCBUS7 to save power for self-

// powered designs

//

// Rev 8 (FT4232H) Extensions (Enabled if Version = 4)

//

0, // non-zero if pull down enabled

0, // non-zero if serial number to be used

0, // non-zero if AL pins have slow slew

0, // non-zero if AL pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if AH pins have slow slew

0, // non-zero if AH pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if BL pins have slow slew

0, // non-zero if BL pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if BH pins have slow slew

0, // non-zero if BH pins are Schmitt input

0, // valid values are 4mA, 8mA, 12mA, 16mA

0, // non-zero if port A uses RI as RS485 TXDEN

0, // non-zero if port B uses RI as RS485 TXDEN

0, // non-zero if port C uses RI as RS485 TXDEN

0, // non-zero if port D uses RI as RS485 TXDEN

0, // non-zero if interface is to use VCP drivers

0, // non-zero if interface is to use VCP drivers

0, // non-zero if interface is to use VCP drivers

0, // non-zero if interface is to use VCP drivers

//

// Rev 9 (FT232H) Extensions (Enabled if Version = 5)

//

0, // non-zero if pull down enabled

0, // non-zero if serial number to be used

0, // non-zero if AC pins have slow slew

0, // non-zero if AC pins are Schmitt input

|  |  |  |
| --- | --- | --- |
| 0, | // | valid values are 4mA, 8mA, 12mA, 16mA |
| 0, | // | non-zero if AD pins have slow slew |
| 0, | // | non-zero if AD pins are Schmitt input |
| 0, | // | valid values are 4mA, 8mA, 12mA, 16mA |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | Cbus Mux control |
| 0, | // | non-zero if interface is 245 FIFO |
| 0, | // | non-zero if interface is 245 FIFO CPU target |
| 0, | // | non-zero if interface is Fast serial |
| 0, | // | non-zero if interface is FT1248 |
| 0, | // | FT1248 clock polarity - clock idle high (1) or |
|  | // | clock idle low (0) |
| 0, | // | FT1248 data is LSB (1) or MSB (0) |
| 0, | // | FT1248 flow control enable |
| 0, | // | non-zero if interface is to use VCP drivers |
| 0 | // | non-zero if using ACBUS7 to save power for |
|  | // | self-powered designs |

};

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); if (ftStatus == FT\_OK) {

ftStatus = FT\_EE\_Program(ftHandle, &ftData); if (ftStatus == FT\_OK) {

// FT\_EE\_Program OK!

}

else {

}

// FT\_EE\_Program FAILED!

FT\_Close(ftHandle);

}

## FT\_EE\_ProgramEx

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Program the EEPROM and pass strings separately.

### Definition

FT\_STATUS **FT\_EE\_ProgramEx** (FT\_HANDLE *ftHandle*, PFT\_PROGRAM\_DATA *pData*,

char \*Manufacturer, char \*ManufacturerId, char \*Description, char \*SerialNumber)

### Parameters

*ftHandle* Handle of the device.

*pData* Pointer to structure of type FT\_PROGRAM\_DATA.

*\*Manufacturer* Pointer to a null-terminated string containing the manufacturer

name.

*\*ManufacturerId* Pointer to a null-terminated string containing the manufacturer ID.

*\*Description* Pointer to a null-terminated string containing the device description.

*\*SerialNumber* Pointer to a null-terminated string containing the device serial

number.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This variation of the [FT\_EE\_Program](#_bookmark55) function was included to provide support for languages such as LabVIEW where problems can occur when string pointers are contained in a structure.

This function interprets the parameter *pData* as a pointer to a structure of type *FT\_PROGRAM\_DATA* that contains the data to write to the EEPROM. The data is written to EEPROM, then read back and verified.

The string pointer parameters in the *FT\_PROGRAM\_DATA* structure should be allocated as DWORDs to avoid overlapping of parameters. The string parameters are then passed in separately.

If the *SerialNumber* field is NULL, or *SerialNumber* points to a NULL string, a serial number based on the *ManufacturerId* and the current date and time will be generated. The *Manufacturer* string length plus the *Description* string length must be less than or equal to 40 characters.

Note that the DLL must be informed which version of the *FT\_PROGRAM\_DATA* structure is being used. This is done through the *Signature1*, *Signature2* and *Version* elements of the structure. *Signature1* should always be *0x00000000*, *Signature2* should always be *0xFFFFFFFF* and *Version* can be set to use whichever version is required. For compatibility with all current devices *Version* should be set to the latest version of the *FT\_PROGRAM\_DATA* structure which is defined in FTD2XX.h.

If *pData* is NULL, the structure version will default to 0 (original BM series) and the device will be programmed with the default data:

## FT\_EE\_UASize

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Get the available size of the EEPROM user area.

### Definition

FT\_STATUS **FT\_EE\_UASize** (FT\_HANDLE *ftHandle*, LPDWORD *lpdwSize*)

### Parameters

*ftHandle* Handle of the device.

*lpdwSize* Pointer to a DWORD that receives the available size, in bytes, of the EEPROM user area.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

The user area of an FTDI device EEPROM is the total area of the EEPROM that is unused by device configuration information and descriptors. This area is available to the user to store information specific to their application. The size of the user area depends on the length of the *Manufacturer*, *ManufacturerId*, *Description* and *SerialNumber* strings programmed into the EEPROM.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (ftStatus != FT\_OK) {

// FT\_Open FAILED!

}

DWORD EEUA\_Size;

ftStatus = FT\_EE\_UASize(ftHandle, &EEUA\_Size); if (ftStatus == FT\_OK) {

// FT\_EE\_UASize OK

// EEUA\_Size contains the size, in bytes, of the EEUA

}

else {

}

// FT\_EE\_UASize FAILED!

FT\_Close(ftHandle);

## FT\_EE\_UARead

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Read the contents of the EEPROM user area.

### Definition

FT\_STATUS **FT\_EE\_UARead** (FT\_HANDLE *ftHandle*, PUCHAR *pucData*, DWORD *dwDataLen*, LPDWORD lpdwBytesRead)

### Parameters

*ftHandle* Handle of the device.

*pucData* Pointer to a buffer that contains storage for data to be read.

*dwDataLen* Size, in bytes, of buffer that contains storage for the data to be read.

*lpdwBytesRead* Pointer to a DWORD that receives the number of bytes read.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function interprets the parameter *pucData* as a pointer to an array of bytes of size *dwDataLen* that contains storage for the data to be read from the EEPROM user area. The actual number of bytes read is stored in the DWORD referenced by *lpdwBytesRead*.

If *dwDataLen* is less than the size of the EEPROM user area, then *dwDataLen* bytes are read into the buffer. Otherwise, the whole of the EEPROM user area is read into the buffer. The available user area size can be determined by calling [FT\_EE\_UASize](#_bookmark57).

An application should check the function return value and *lpdwBytesRead* when FT\_EE\_UARead returns.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (ftStatus != FT\_OK) {

// FT\_Open FAILED!

}

unsigned char Buffer[64]; DWORD BytesRead;

ftStatus = FT\_EE\_UARead(ftHandle, Buffer, 64, &BytesRead); if (ftStatus == FT\_OK) {

// FT\_EE\_UARead OK

// User Area data stored in Buffer

// Number of bytes read from EEUA stored in BytesRead

}

else {

}

// FT\_EE\_UARead FAILED!

FT\_Close(ftHandle);

## FT\_EE\_UAWrite

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Write data into the EEPROM user area.

### Definition

FT\_STATUS **FT\_EE\_UAWrite** (FT\_HANDLE *ftHandle*, PUCHAR *pucData*, DWORD *dwDataLen*)

### Parameters

*ftHandle* Handle of the device.

*pucData* Pointer to a buffer that contains the data to be written.

*dwDataLen* Size, in bytes, of buffer that contains storage for the data to be read.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function interprets the parameter *pucData* as a pointer to an array of bytes of size *dwDataLen* that contains the data to be written to the EEPROM user area. It is a programming error for *dwDataLen* to be greater than the size of the EEPROM user area. The available user area size can be determined by calling [FT\_EE\_UASize](#_bookmark57).

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (ftStatus != FT\_OK) {

// FT\_Open FAILED!

}

char \*buffer = (char \*) "Hello, World";

ftStatus = FT\_EE\_UAWrite(ftHandle, (unsigned char\*)buffer, 12); if(ftStatus != FT\_OK) {

// FT\_EE\_UAWRITE failed

}

else {

}

// FT\_EE\_UAWRITE failed

FT\_Close(ftHandle);

## FT\_EEPROM\_Read

### Supported Operating Systems

Windows (XP and later) Linux

### Summary

Read data from the EEPROM, this command will work for all existing FTDI chipset, and must be used for the FT-X series.

### Definition

FT\_STATUS **FT\_EEPROM\_Read**(FT\_HANDLE ftHandle, void \*eepromData, DWORD eepromDataSize,

char \*Manufacturer, char \*ManufacturerId, char \*Description, char \*SerialNumber);

### Parameters

*ftHandle* Handle of the device.

*\*eepromData* Pointer to a buffer that contains the data to be read. Note: This structure is different for each device type.

*epromDataSize* Size of the eepromData buffer that contains storage for the data to be

read.

*\*Manufacturer* Pointer to a null-terminated string containing the manufacturer

name

*\*ManufacturerId* Pointer to a null-terminated string containing the manufacturer ID.

*\*Description* Pointer to a null-terminated string containing the device description.

*\*SerialNumber* Pointer to a null-terminated string containing the device serial

number.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function interprets the parameter *\*eepromDATA* as a pointer to a structure matching the device type being accessed e.g.

*PFT\_EEPROM\_232B* is the structure for FT2xxB devices. *PFT\_EEPROM\_2232* is the structure for FT2232D devices. *PFT\_EEPROM\_232R* is the structure for FT232R devices. *PFT\_EEPROM\_2232H* is the structure for FT2232H devices. *PFT\_EEPROM\_4232H* is the structure for FT4232H devices. *PFT\_EEPROM\_232H* is the structure for FT232H devices.

*PFT\_EEPROM\_X\_SERIES* is the structure for FT2xxX devices.

The function does not perform any checks on buffer sizes, so the buffers passed in the *eepromDATA*

structure must be big enough to accommodate their respective strings (including null terminators).

The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the *Manufacturer* string length plus the *Description* string length is less than or equal to 40 characters.

Note that the DLL must be informed which version of the *eepromDATA* structure is being used. This is done through the *PFT\_EEPROM\_HEADER* structure. The first element of this structure is deviceType and may be FT\_DEVICE\_BM, FT\_DEVICE\_AM, FT\_DEVICE\_2232C, FT\_DEVICE\_232R, FT\_DEVICE\_2232H, FT\_DEVICE\_4232H, FT\_DEVICE\_232H, or FT\_DEVICE\_X\_SERIES as defined in FTD2XX.h.

**Example** FT\_HANDLE fthandle; FT\_STATUS status;

char Manufacturer[64]; char ManufacturerId[64]; char Description[64]; char SerialNumber[64];

FT\_EEPROM\_HEADER ft\_eeprom\_header;

ft\_eeprom\_header.deviceType = FT\_DEVICE\_2232H; // FTxxxx device type to be accessed FT\_EEPROM\_2232H ft\_eeprom\_2232h;

ft\_eeprom\_2232h.common = ft\_eeprom\_header; ft\_eeprom\_2232h.common.deviceType = FT\_DEVICE\_2232H;

status = FT\_Open(0, &fthandle); if(status != FT\_OK)

printf("open status not ok %d\n", status);

status = FT\_EEPROM\_Read(fthandle,&ft\_eeprom\_2232h, sizeof(ft\_eeprom\_2232h), Manufacturer,ManufacturerId, Description, SerialNumber);

if (status != FT\_OK)

printf("EEPROM\_Read status not ok %d\n", status);

else

{

}

printf("VendorID = 0x%04x\n", ft\_eeprom\_2232h.common.VendorId); printf("ProductID = 0x%04x\n", ft\_eeprom\_2232h.common.ProductId);

…

…

FT\_Close(fthandle);

## FT\_EEPROM\_Program

### Supported Operating Systems

Windows (XP and later) Linux

### Summary

Write data into the EEPROM, this command will work for all existing FTDI chipset, and must be used for the FT-X series.

### Definition

FT\_STATUS **FT\_EEPROM\_Program**(FT\_HANDLE ftHandle, void \*eepromData, DWORD eepromDataSize,

char \*Manufacturer, char \*ManufacturerId, char \*Description, char \*SerialNumber);

### Parameters

*ftHandle* Handle of the device.

*\*eepromData* Pointer to a buffer that contains the data to be written. Note: This structure is different for each device type.

*epromDataSize* Size of the eepromData buffer that contains storage for the data to be

written.

*\*Manufacturer* Pointer to a null-terminated string containing the manufacturer

name

*\*ManufacturerId* Pointer to a null-terminated string containing the manufacturer ID.

*\*Description* Pointer to a null-terminated string containing the device description.

*\*SerialNumber* Pointer to a null-terminated string containing the device serial

number.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function interprets the parameter *\*eepromDATA* as a pointer to a structure matching the device type being accessed e.g.

*PFT\_EEPROM\_232B* is the structure for FT2xxB devices. *PFT\_EEPROM\_2232* is the structure for FT2232D devices. *PFT\_EEPROM\_232R* is the structure for FT232R devices. *PFT\_EEPROM\_2232H* is the structure for FT2232H devices. *PFT\_EEPROM\_4232H* is the structure for FT4232H devices. *PFT\_EEPROM\_232H* is the structure for FT232H devices.

*PFT\_EEPROM\_X\_SERIES* is the structure for FT2xxX devices.

The function does not perform any checks on buffer sizes, so the buffers passed in the *eepromDATA*

structure must be big enough to accommodate their respective strings (including null terminators).

The sizes shown in the following example are more than adequate and can be rounded down if necessary. The restriction is that the *Manufacturer* string length plus the *Description* string length is less than or equal to 40 characters.

Note that the DLL must be informed which version of the *eepromDATA* structure is being used. This is done through the *PFT\_EEPROM\_HEADER* structure. The first element of this structure is deviceType and may be FT\_DEVICE\_BM, FT\_DEVICE\_AM, FT\_DEVICE\_2232C, FT\_DEVICE\_232R, FT\_DEVICE\_2232H, FT\_DEVICE\_4232H, FT\_DEVICE\_232H, or FT\_DEVICE\_X\_SERIES as defined in FTD2XX.h.

**Example** FT\_HANDLE fthandle; FT\_STATUS status;

char Manufacturer[64]; char ManufacturerId[64]; char Description[64]; char SerialNumber[64];

FT\_EEPROM\_HEADER ft\_eeprom\_header;

ft\_eeprom\_header.deviceType = FT\_DEVICE\_2232H; // FTxxxx device type to be accessed FT\_EEPROM\_2232H ft\_eeprom\_2232h;

ft\_eeprom\_2232h.common = ft\_eeprom\_header; ft\_eeprom\_2232h.common.deviceType = FT\_DEVICE\_2232H;

status = FT\_Open(0, &fthandle);

if(status != FT\_OK)

printf("open status not ok %d\n", status);

status = FT\_EEPROM\_Read(fthandle,&ft\_eeprom\_2232h, sizeof(ft\_eeprom\_2232h), Manufacturer, ManufacturerId, Description, SerialNumber);

strcpy\_s(SerialNumber, (const char \*) "FT000001");

status = FT\_EEPROM\_Program(fthandle,&ft\_eeprom\_2232h, sizeof(ft\_eeprom\_2232h), Manufacturer, ManufacturerId, Description, SerialNumber);

FT\_Close(fthandle);

# Extended API Functions

The extended API functions do not apply to FT8U232AM or FT8U245AM devices. FTDI’s other USB-UART and USB-FIFO ICs (the FT2232H, FT4232H, FT232R, FT245R, FT2232, FT232B and FT245B) do support

these functions. Note that there is device dependence in some of these functions.

## FT\_SetLatencyTimer

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Set the latency timer value.

### Definition

FT\_STATUS **FT\_SetLatencyTimer** (FT\_HANDLE *ftHandle*, UCHAR *ucTimer*)

### Parameters

*ftHandle* Handle of the device.

*ucTimer* Required value, in milliseconds, of latency timer. Valid range is 2 – 255.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

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.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; UCHAR LatencyTimer = 10;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetLatencyTimer(ftHandle, LatencyTimer); if (ftStatus == FT\_OK) {

// LatencyTimer set to 10 milliseconds

}

else {

}

// FT\_SetLatencyTimer FAILED!

FT\_Close(ftHandle);

## FT\_GetLatencyTimer

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Get the current value of the latency timer.

### Definition

FT\_STATUS **FT\_GetLatencyTimer** (FT\_HANDLE *ftHandle*, PUCHAR *pucTimer*)

### Parameters

*ftHandle* Handle of the device.

*pucTimer* Pointer to unsigned char to store latency timer value.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

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.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; UCHAR LatencyTimer;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_GetLatencyTimer(ftHandle, &LatencyTimer); if (ftStatus == FT\_OK) {

// LatencyTimer contains current value

}

else {

}

// FT\_GetLatencyTimer FAILED!

FT\_Close(ftHandle);

## FT\_SetBitMode

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Enables different chip modes.

### Definition

FT\_STATUS **FT\_SetBitMode** (FT\_HANDLE *ftHandle*, UCHAR *ucMask*, UCHAR *ucMode*)

### Parameters

*ftHandle* Handle of the device.

*ucMask* Required value for bit mode mask. This sets up which bits are inputs and outputs. A bit value of 0 sets the corresponding pin to an input, a bit value of 1 sets the corresponding pin to an output.

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.

*ucMode* Mode value. Can be one of the following: 0x0 = Reset

0x1 = Asynchronous Bit Bang

0x2 = MPSSE (FT2232, FT2232H, FT4232H and FT232H devices only)

0x4 = Synchronous Bit Bang (FT232R, FT245R, FT2232, FT2232H, FT4232H and FT232H devices only) 0x8 = MCU Host Bus Emulation Mode (FT2232, FT2232H, FT4232H and FT232H devices only)

0x10 = Fast Opto-Isolated Serial Mode (FT2232, FT2232H, FT4232H and FT232H devices only) 0x20 = CBUS Bit Bang Mode (FT232R and FT232H devices only)

0x40 = Single Channel Synchronous 245 FIFO Mode (FT2232H and FT232H devices only)

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

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.

### Example

FT\_HANDLE ftHandle; FT\_STATUS ftStatus; UCHAR Mask = 0xff;

UCHAR Mode = 1; // Set asynchronous bit-bang mode

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetBitMode(ftHandle, Mask, Mode); if (ftStatus == FT\_OK) {

// 0xff written to device

}

else {

}

// FT\_SetBitMode FAILED!

FT\_Close(ftHandle);

## FT\_GetBitMode

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Gets the instantaneous value of the data bus.

### Definition

FT\_STATUS **FT\_GetBitMode** (FT\_HANDLE *ftHandle*, PUCHAR *pucMode*)

### Parameters

*ftHandle* Handle of the device.

*pucMode* Pointer to unsigned char to store the instantaneous data bus value.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

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.

### Example

FT\_HANDLE ftHandle;

UCHAR BitMode;

FT\_STATUS ftStatus;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_GetBitMode(ftHandle, &BitMode); if (ftStatus == FT\_OK) {

// BitMode contains current value

}

else {

}

// FT\_GetBitMode FAILED!

FT\_Close(ftHandle);

## FT\_SetUSBParameters

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Set the USB request transfer size.

### Definition

FT\_STATUS **FT\_SetUSBParameters** (FT\_HANDLE *ftHandle*, DWORD *dwInTransferSize*, DWORD dwOutTransferSize)

### Parameters

*ftHandle* Handle of the device.

*dwInTransferSize* Transfer size for USB IN request.

*dwOutTransferSize* Transfer size for USB OUT request.

### Return Value

FT\_OK if successful, otherwise the return value is an FT error code.

### Remarks

This function can be used to change the transfer sizes from the default transfer size of 4096 bytes to better suit the application requirements. Transfer sizes must be set to a multiple of 64 bytes between 64 bytes and 64k bytes.

When FT\_SetUSBParameters is called, the change comes into effect immediately and any data that was held in the driver at the time of the change is lost.

Note that, at present, only *dwInTransferSize* is supported.

### Example

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus;

DWORD InTransferSize = 16384;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus != FT\_OK) {

// FT\_Open failed return;

}

ftStatus = FT\_SetUSBParameters(ftHandle, InTransferSize, 0); if (ftStatus == FT\_OK) {

// In transfer size set to 16 Kbytes

}

else {

}

// FT\_SetUSBParameters FAILED!

FT\_Close(ftHandle);

# FT-Win32 API Functions

The functions in this section are supplied to ease porting from a Win32 serial port application. These functions are supported under non-Windows platforms to assist with porting existing applications from Windows. Note that classic D2XX functions and the Win32 D2XX functions should not be mixed unless stated.

## FT\_W32\_CreateFile

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Opens the specified device and return a handle which will be used for subsequent accesses. The device

can be specified by its serial number, device description, or location. This function must be used if overlapped I/O is required.

### Definition

FT\_HANDLE **FT\_W32\_CreateFile** (PVOID *pvArg1*, DWORD *dwAccess*, DWORD *dwShareMode*,

LPSECURITY\_ATTRIBUTES *lpSecurityAttributes*, DWORD *dwCreate*, DWORD dwAttrsAndFlags, HANDLE hTemplate)

### Parameters

*pvArg1* Meaning depends on the value of *dwAttrsAndFlags*. Can be a pointer to a null terminated string that contains the description or serial number of the device, or can be the location of the device. These values can be obtained from the [FT\_CreateDeviceInfoList](#_bookmark6), [FT\_GetDeviceInfoDetail](#_bookmark8) or [FT\_ListDevices](#_bookmark9) functions.

*dwAccess* Type of access to the device. Access can be GENERIC\_READ, GENERIC\_WRITE or both. Ignored in Linux.

*dwShareMode* How the device is shared. This value must be set to 0.

*lpSecurityAttributes* This parameter has no effect and should be set to NULL.

*dwCreate* This parameter must be set to OPEN\_EXISTING. Ignored in Linux.

*dwAttrsAndFlags* File attributes and flags. This parameter is a combination of FILE\_ATTRIBUTE\_NORMAL, FILE\_FLAG\_OVERLAPPED if overlapped I/O is used, [FT\_OPEN\_BY\_SERIAL\_NUMBER](#_bookmark93) if *lpszName* is the device’s serial number, and [FT\_OPEN\_BY\_DESCRIPTION](#_bookmark93) if *lpszName* is the device’s description.

*hTemplate* This parameter must be NULL.

### Return Value

If the function is successful, the return value is a handle.

If the function is unsuccessful, the return value is the Win32 error code INVALID\_HANDLE\_VALUE.

### Remarks

The meaning of pvArg1 depends on dwAttrsAndFlags: if [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93) or [*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) is set in dwAttrsAndFlags, pvArg1 contains a pointer to a null terminated string that contains the device's serial number or description; if [*FT\_OPEN\_BY\_LOCATION*](#_bookmark93) is set in dwAttrsAndFlags, pvArg1 is interpreted as a value of type long that contains the location ID of the device.

dwAccess can be GENERIC\_READ, GENERIC\_WRITE or both; dwShareMode must be set to 0; lpSecurityAttributes must be set to NULL; dwCreate must be set to OPEN\_EXISTING; dwAttrsAndFlags is a combination of FILE\_ATTRIBUTE\_NORMAL, FILE\_FLAG\_OVERLAPPED if overlapped I/O is used, [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93) or [*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) or [*FT\_OPEN\_BY\_LOCATION*](#_bookmark93); hTemplate

must be NULL.

Note that Linux, Mac OS X and Windows CE do not support overlapped IO or location IDs.

### Examples

The examples that follow use these variables.

FT\_STATUS ftStatus; FT\_HANDLE ftHandle; char Buf[64];

1. Open a device for overlapped I/O using its serial number

ftStatus = FT\_ListDevices(0,Buf,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_SERIAL\_NUMBER);

ftHandle = FT\_W32\_CreateFile((LPCTSTR) Buf,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED | FT\_OPEN\_BY\_SERIAL\_NUMBER,

0);

if (ftHandle == INVALID\_HANDLE\_VALUE)

; // FT\_W32\_CreateDevice failed

1. Open a device for non-overlapped I/O using its description

ftStatus = FT\_ListDevices(0,Buf,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_DESCRIPTION);

ftHandle = FT\_W32\_CreateFile((LPCTSTR) Buf,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL | FT\_OPEN\_BY\_DESCRIPTION, 0);

if (ftHandle == INVALID\_HANDLE\_VALUE)

; // FT\_W32\_CreateDevice failed

1. Open a device for non-overlapped I/O using its location

long locID;

ftStatus = FT\_ListDevices(0,&locID,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_LOCATION); ftHandle = FT\_W32\_CreateFile((LPCTSTR) locID,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL | FT\_OPEN\_BY\_LOCATION, 0);

if (ftHandle == INVALID\_HANDLE\_VALUE)

; // FT\_W32\_CreateDevice failed

## FT\_W32\_CloseHandle

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Close the specified device handle.

### Definition

BOOL **FT\_W32\_CloseHandle** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

This example shows how to close a device after opening it for non-overlapped I/O using its description.

FT\_STATUS ftStatus; FT\_HANDLE ftHandle; char Buf[64];

ftStatus = FT\_ListDevices(0,Buf,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_DESCRIPTION); ftHandle = FT\_W32\_CreateFile((LPCTSTR) Buf,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL | FT\_OPEN\_BY\_DESCRIPTION, 0);

if (ftHandle == INVALID\_HANDLE\_VALUE){

// FT\_W32\_CreateDevice failed

}

else {

}

// FT\_W32\_CreateFile OK, so do some work, and eventually ... FT\_W32\_CloseHandle(ftHandle);

## FT\_W32\_ReadFile

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later)

### Summary

Read data from the device.

### Definition

BOOL **FT\_W32\_ReadFile** (FT\_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*,

LPDWORD lpdwBytesReturned, LPOVERLAPPED lpOverlapped)

### Parameters

*ftHandle* Handle of the device.

*lpBuffer* Pointer to a buffer that receives the data from the device.

*dwBytesToRead* Number of bytes to read from the device.

*lpdwBytesReturned* Pointer to a variable that receives the number of bytes read from the device.

*lpOverlapped* Pointer to an overlapped structure.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

This function supports both non-overlapped and overlapped I/O, except under Linux, Mac OS X and

Windows CE where only non-overlapped IO is supported.

Non-overlapped I/O

The parameter, *lpOverlapped,* must be NULL for non-overlapped I/O.

This function always returns the number of bytes read in *lpdwBytesReturned*.

This function does not return until *dwBytesToRead* have been read into the buffer. The number of bytes in the receive queue can be determined by calling [FT\_GetStatus](#_bookmark31) or [FT\_GetQueueStatus](#_bookmark26), and passed as *dwBytesToRead* so that the function reads the device and returns immediately.

When a read timeout has been setup in a previous call to [FT\_W32\_SetCommTimeouts](#_bookmark79), this function returns when the timer expires or *dwBytesToRead* have been read, whichever occurs first. If a timeout occurred, any available data is read into *lpBuffer* and the function returns a non-zero value.

An application should use the function return value and *lpdwBytesReturned* when processing the buffer. If the return value is non-zero and *lpdwBytesReturned* is equal to *dwBytesToRead* then the function has completed normally. If the return value is non-zero and *lpdwBytesReturned* is less then *dwBytesToRead* then a timeout has occurred, and the read request has been partially completed. Note that if a timeout occurred and no data was read, the return value is still non-zero.

A return value of *FT\_IO\_ERROR* suggests an error in the parameters of the function, or a fatal error like USB disconnect has occurred.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped,* must point to an initialized OVERLAPPED structure.

If there is enough data in the receive queue to satisfy the request, the request completes immediately and the return code is non-zero. The number of bytes read is returned in *lpdwBytesReturned*.

If there is not enough data in the receive queue to satisfy the request, the request completes immediately, and the return code is zero, signifying an error. An application should call [FT\_W32\_GetLastError](#_bookmark87) to get the cause of the error. If the error code is ERROR\_IO\_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling [FT\_W32\_GetOverlappedResult](#_bookmark73).

If successful, the number of bytes read is returned in *lpdwBytesReturned*.

### Example

1. This example shows how to read 256 bytes from the device using non-overlapped I/O.

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile for non-overlapped i/o FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

char Buf[256];

DWORD dwToRead = 256; DWORD dwRead;

DWORD osRead;

if (FT\_W32\_ReadFile(ftHandle, Buf, dwToRead, &dwRead, (LPOVERLAPPED) &osRead)) { if (dwToRead == dwRead){

// FT\_W32\_ReadFile OK

}

else{

}

else{

}

// FT\_W32\_ReadFile timeout

// FT\_W32\_ReadFile failed

}

1. This example shows how to read 256 bytes from the device using overlapped I/O. FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); char Buf[256];

DWORD dwToRead = 256; DWORD dwRead;

OVERLAPPED osRead = { 0 };

osRead.hEvent = CreateEvent (NULL, FALSE, FALSE, NULL);

if (!FT\_W32\_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osRead)) { if (FT\_W32\_GetLastError(ftHandle) == ERROR\_IO\_PENDING) {

// write is delayed so do some other stuff until ...

if (!FT\_W32\_GetOverlappedResult(ftHandle, &osRead, &dwRead, FALSE)){

// error

}

else {

if (dwToRead == dwRead){

// FT\_W32\_ReadFile OK

}

}

}

else {

}

else{

}

// FT\_W32\_ReadFile timeout

// FT\_W32\_ReadFile OK

}

CloseHandle (osRead.hEvent);

## FT\_W32\_WriteFile

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Write data to the device.

### Definition

BOOL **FT\_W32\_WriteFile** (FT\_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*,

LPDWORD lpdwBytesWritten, LPOVERLAPPED lpOverlapped)

### Parameters

*ftHandle* Handle of the device.

*lpBuffer* Pointer to the buffer that contains the data to write to the device.

*dwBytesToWrite* Number of bytes to be written to the device.

*lpdwBytesWritten* Pointer to a variable that receives the number of bytes written to the device.

*lpOverlapped* Pointer to an overlapped structure.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

This function supports both non-overlapped and overlapped I/O, except under Linux, Mac OS X and

Windows CE where only non-overlapped IO is supported. Non-overlapped I/O

The parameter, *lpOverlapped,* must be NULL for non-overlapped I/O.

This function always returns the number of bytes written in *lpdwBytesWritten*.

This function does not return until *dwBytesToWrite* have been written to the device.

When a write timeout has been setup in a previous call to [FT\_W32\_SetCommTimeouts](#_bookmark79), this function returns when the timer expires or *dwBytesToWrite* have been written, whichever occurs first. If a timeout occurred, *lpdwBytesWritten* contains the number of bytes actually written, and the function returns a

non-zero value.

An application should always use the function return value and *lpdwBytesWritten*. If the return value is non-zero and *lpdwBytesWritten* is equal to *dwBytesToWrite* then the function has completed normally. If the return value is non-zero and *lpdwBytesWritten* is less then *dwBytesToWrite* then a timeout has occurred, and the write request has been partially completed. Note that if a timeout occurred and no data was written, the return value is still non-zero.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in

which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped,* must point to an initialized OVERLAPPED structure.

This function completes immediately, and the return code is zero, signifying an error. An application should call [FT\_W32\_GetLastError](#_bookmark87) to get the cause of the error. If the error code is ERROR\_IO\_PENDING, the overlapped operation is still in progress, and the application can perform other processing.

Eventually, the application checks the result of the overlapped request by calling [FT\_W32\_GetOverlappedResult](#_bookmark73).

If successful, the number of bytes written is returned in *lpdwBytesWritten*.

### Example

1. This example shows how to write 128 bytes to the device using non-overlapped I/O.

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

// setup by FT\_W32\_CreateFile for non-overlapped i/o char Buf[128]; // contains data to write to the device DWORD dwToWrite = 128;

DWORD dwWritten;

DWORD osWrite;

if (FT\_W32\_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, (LPOVERLAPPED) &osWrite)) { if (dwToWrite == dwWritten){

// FT\_W32\_WriteFile OK

}

else{

}

else{

}

// FT\_W32\_WriteFile timeout

// FT\_W32\_WriteFile failed

}

1. This example shows how to write 128 bytes to the device using overlapped I/O. FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

// setup by FT\_W32\_CreateFile for overlapped i/o

char Buf[128]; // contains data to write to the device DWORD dwToWrite = 128;

DWORD dwWritten; OVERLAPPED osWrite = { 0 };

if (!FT\_W32\_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) { if (FT\_W32\_GetLastError(ftHandle) == ERROR\_IO\_PENDING) {

// write is delayed so do some other stuff until ...

if (!FT\_W32\_GetOverlappedResult(ftHandle, &osWrite, &dwWritten, FALSE)){

// error

}

else {

if (dwToWrite == dwWritten){

// FT\_W32\_WriteFile OK

}

else{

}

}

}

// FT\_W32\_WriteFile timeout

}

else {

}

// FT\_W32\_WriteFIle OK

## FT\_W32\_GetOverlappedResult

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Gets the result of an overlapped operation.

### Definition

BOOL **FT\_W32\_GetOverlappedResult** (FT\_HANDLE *ftHandle*, LPOVERLAPPED *lpOverlapped*,

LPDWORD lpdwBytesTransferred, BOOL bWait)

### Parameters

*ftHandle* Handle of the device.

*lpOverlapped* Pointer to an overlapped structure.

*lpdwBytesTransferred* Pointer to a variable that receives the number of bytes transferred during the overlapped operation.

*bWait* Set to TRUE if the function does not return until the operation has been completed.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

This function is used with overlapped I/O and so is not supported in Linux, Mac OS X or Windows CE. For

a description of its use, see [FT\_W32\_ReadFile](#_bookmark71) and [FT\_W32\_WriteFile](#_bookmark72).

## FT\_W32\_EscapeCommFunction

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Perform an extended function.

### Definition

BOOL **FT\_W32\_EscapeCommFunction** (FT\_HANDLE *ftHandle*, DWORD *dwFunc*)

### Parameters

*ftHandle* Handle of the device.

*dwFunc* The extended function to perform can be one of the following values:

CLRDTR – Clear the DTR signal CLRRTS – Clear the RTS signal SETDTR – Set the DTR signal SETRTS – Set the RTS signal SETBREAK – Set the BREAK condition

CLRBREAK – Clear the BREAK condition

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_W32\_EscapeCommFunction(ftHandle,CLRDTR); // Clear the DTR signal FT\_W32\_EscapeCommFunction(ftHandle,SETRTS); // Set the RTS signal

## FT\_W32\_GetCommModemStatus

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function gets the current modem control value.

### Definition

BOOL **FT\_W32\_GetCommModemStatus** (FT\_HANDLE *ftHandle*, LPDWORD *lpdwStat*)

### Parameters

*ftHandle* Handle of the device.

*lpdwStat* Pointer to a variable to contain modem control value. The modem control value can be a combination of the following:

MS\_CTS\_ON – Clear To Send (CTS) is on MS\_DSR\_ON – Data Set Ready (DSR) is on

MS\_RING\_ON – Ring Indicator (RI) is on

MS\_RLSD\_ON – Receive Line Signal Detect (RLSD) is on

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); DWORD dwStatus;

if (FT\_W32\_GetCommModemStatus(ftHandle,(LPDWORD)&dwStatus)) {

// FT\_W32\_GetCommModemStatus ok if (dwStatus & MS\_CTS\_ON)

; // CTS is on

if (dwStatus & MS\_DSR\_ON)

; // DSR is on

if (dwStatus & MS\_RING\_ON)

; // RI is on

if (dwStatus & MS\_RLSD\_ON)

; // RLSD is on

}

else

; // FT\_W32\_GetCommModemStatus failed

## FT\_W32\_SetupComm

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the read and write buffers.

### Definition

BOOL **FT\_W32\_SetupComm** (FT\_HANDLE *ftHandle*, DWORD *dwReadBufferSize*,

DWORD dwWriteBufferSize)

### Parameters

*ftHandle* Handle of the device.

*dwReadBufferSize* Length, in bytes, of the read buffer. *dwWriteBufferSize* Length, in bytes, of the write buffer. **Return Value**

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

### Remarks

This function has no effect. It is the responsibility of the driver to allocate sufficient storage for I/O requests.

## FT\_W32\_SetCommState

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the state of the device according to the contents of a device control block (DCB).

### Definition

BOOL **FT\_W32\_SetCommState** (FT\_HANDLE *ftHandle*, LPFTDCB *lpftDcb*)

### Parameters

*ftHandle* Handle of the device.

*lpftDcb* Pointer to an FTDCB structure.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);// setup by FT\_W32\_CreateFile FTDCB ftDCB;

if (FT\_W32\_GetCommState(ftHandle,&ftDCB)) {

// FT\_W32\_GetCommState ok, device state is in ftDCB ftDCB.BaudRate = 921600; // Change the baud rate

if (FT\_W32\_SetCommState(ftHandle,&ftDCB))

; // FT\_W32\_SetCommState ok

}

else

else

; // FT\_W32\_SetCommState failed

; // FT\_W32\_GetCommState failed

## FT\_W32\_GetCommState

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later)

Windows CE (4.2 and later)

### Summary

This function gets the current device state.

### Definition

BOOL **FT\_W32\_GetCommState** (FT\_HANDLE *ftHandle*, LPFTDCB *lpftDcb*)

### Parameters

*ftHandle* Handle of the device.

*lpftDcb* Pointer to an FTDCB structure.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

The current state of the device is returned in a device control block.

### Example

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); FTDCB ftDCB; if (FT\_W32\_GetCommState(ftHandle,&ftDCB))

; // FT\_W32\_GetCommState ok, device state is in ftDCB

else

; // FT\_W32\_GetCommState failed

## FT\_W32\_SetCommTimeouts

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function sets the timeout parameters for I/O requests.

### Definition

BOOL **FT\_W32\_SetCommTimeouts** (FT\_HANDLE *ftHandle*, LPFTTIMEOUTS *lpftTimeouts*)

### Parameters

*ftHandle* Handle of the device.

*lpftTimeouts* Pointer to an FTTIMEOUTS structure to store timeout information.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

Timeouts are calculated using the information in the FTTIMEOUTS structure.

For read requests, the number of bytes to be read is multiplied by the total timeout multiplier, and added to the total timeout constant. So, if TS is an FTTIMEOUTS structure and the number of bytes to read is dwToRead, the read timeout, rdTO, is calculated as follows.

rdTO = (dwToRead \* TS.ReadTotalTimeoutMultiplier) + TS.ReadTotalTimeoutConstant

For write requests, the number of bytes to be written is multiplied by the total timeout multiplier, and added to the total timeout constant. So, if TS is an FTTIMEOUTS structure and the number of bytes to write is dwToWrite, the write timeout, wrTO, is calculated as follows.

wrTO = (dwToWrite \* TS.WriteTotalTimeoutMultiplier) + TS.WriteTotalTimeoutConstant

Linux and Mac OS X currently ignore the ReadIntervalTimeout, ReadTotalTimeoutMultiplier and WriteTotalTimeoutMultiplier.

### Example

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); FTTIMEOUTS ftTS;

ftTS.ReadIntervalTimeout = 0;

ftTS.ReadTotalTimeoutMultiplier = 0;

ftTS.ReadTotalTimeoutConstant = 100;

ftTS.WriteTotalTimeoutMultiplier = 0;

ftTS.WriteTotalTimeoutConstant = 200;

if (FT\_W32\_SetCommTimeouts(ftHandle,&ftTS))

; // FT\_W32\_SetCommTimeouts OK

else

; // FT\_W32\_SetCommTimeouts failed

## FT\_W32\_GetCommTimeouts

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function gets the current read and write request timeout parameters for the specified device.

### Definition

BOOL **FT\_W32\_GetCommTimeouts** (FT\_HANDLE *ftHandle*, LPFTTIMEOUTS *lpftTimeouts*)

### Parameters

*ftHandle* Handle of the device.

*lpftTimeouts* Pointer to an FTTIMEOUTS structure to store timeout information.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

For an explanation of how timeouts are used, see [FT\_W32\_SetCommTimeouts](#_bookmark79).

### Example

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); FTTIMEOUTS ftTS;

if (FT\_W32\_GetCommTimeouts(ftHandle,&ftTS))

; // FT\_W32\_GetCommTimeouts OK

else

; // FT\_W32\_GetCommTimeouts failed

## FT\_W32\_SetCommBreak

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Puts the communications line in the BREAK state.

### Definition

BOOL **FT\_W32\_SetCommBreak** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (!FT\_W32\_SetCommBreak(ftHandle))

; // FT\_W32\_SetCommBreak failed

else

; // FT\_W32\_SetCommBreak OK

## FT\_W32\_ClearCommBreak

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Puts the communications line in the non-BREAK state.

### Definition

BOOL **FT\_W32\_ClearCommBreak** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (!FT\_W32\_ClearCommBreak(ftHandle)){

// FT\_W32\_ClearCommBreak failed

}

else{

}

// FT\_W32\_ClearCommBreak OK

## FT\_W32\_SetCommMask

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function specifies events that the device has to monitor.

### Definition

BOOL **FT\_W32\_SetCommMask** (FT\_HANDLE *ftHandle*, DWORD *dwMask*)

### Parameters

*ftHandle* Handle of the device.

*dwMask* Mask containing events that the device has to monitor. This can be a combination of the following:

EV\_BREAK – BREAK condition detected EV\_CTS – Change in Clear To Send (CTS) EV\_DSR – Change in Data Set Ready (DSR) EV\_ERR – Error in line status

EV\_RING – Change in Ring Indicator (RI)

EV\_RLSD – Change in Receive Line Signal Detect (RLSD) EV\_RXCHAR – Character received

EV\_RXFLAG – Event character received EV\_TXEMPTY – Transmitter empty

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

This function specifies the events that the device should monitor. An application can call the function

[FT\_W32\_WaitCommEvent](#_bookmark85) to wait for an event to occur.

### Example

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

DWORD dwMask = EV\_CTS | EV\_DSR;

if (!FT\_W32\_SetCommMask(ftHandle,dwMask))

; // FT\_W32\_SetCommMask failed

else

; // FT\_W32\_SetCommMask OK

## FT\_W32\_GetCommMask

**Supported Operating Systems** Windows (2000 and later) **Summary**

Retrieves the events that are currently being monitored by a device.

### Definition

BOOL **FT\_W32\_GetCommMask** (FT\_HANDLE *ftHandle*, LPDWORD *lpdwEventMask*)

### Parameters

*ftHandle* Handle of the device.

*lpdwEventMask* Pointer to a location that receives a mask that contains the events

that are currently enabled. This parameter can be one or more of the following values:

EV\_BREAK – BREAK condition detected EV\_CTS – Change in Clear To Send (CTS) EV\_DSR – Change in Data Set Ready (DSR)

### Return Value

EV\_ERR – Error in line status

EV\_RING – Change in Ring Indicator (RI)

EV\_RLSD – Change in Receive Line Signal Detect (RLSD) EV\_RXCHAR – Character received

EV\_RXFLAG – Event character received EV\_TXEMPTY – Transmitter empty

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

This function returns events currently being monitored by the device. Event monitoring for these events

is enabled by the [FT\_W32\_SetCommMask](#_bookmark83) function.

### Example

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); DWORD dwMask;

if (!FT\_W32\_GetCommMask(ftHandle,&dwMask))

; // FT\_W32\_GetCommMask failed

else

; // FT\_W32\_GetCommMask OK

## FT\_W32\_WaitCommEvent

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function waits for an event to occur.

### Definition

BOOL **FT\_W32\_WaitCommEvent** (FT\_HANDLE *ftHandle*, LPDWORD *lpdwEvent*,

LPOVERLAPPED *lpOverlapped*)

### Parameters

*ftHandle* Handle of the device.

*lpdwEvent* Pointer to a location that receives a mask that contains the events that occurred.

*lpOverlapped* Pointer to an overlapped structure.

### Return Value

If the function is successful, the return value is nonzero.

If the function is unsuccessful, the return value is zero.

### Remarks

This function supports both non-overlapped and overlapped I/O, except under Windows CE and Linux where only non-overlapped IO is supported.

Non-overlapped I/O

The parameter, *lpOverlapped,* must be NULL for non-overlapped I/O.

This function does not return until an event that has been specified in a call to [FT\_W32\_SetCommMask](#_bookmark83) has occurred. The events that occurred and resulted in this function returning are stored in *lpdwEvent*.

Overlapped I/O

When the device has been opened for overlapped I/O, an application can issue a request and perform some additional work while the request is pending. This contrasts with the case of non-overlapped I/O in which the application issues a request and receives control again only after the request has been completed.

The parameter, *lpOverlapped,* must point to an initialized OVERLAPPED structure.

This function does not return until an event that has been specified in a call to [FT\_W32\_SetCommMask](#_bookmark83) has occurred.

If an event has already occurred, the request completes immediately, and the return code is non-zero. The events that occurred are stored in *lpdwEvent*.

If an event has not yet occurred, the request completes immediately, and the return code is zero, signifying an error. An application should call [FT\_W32\_GetLastError](#_bookmark87) to get the cause of the error. If the error code is ERROR\_IO\_PENDING, the overlapped operation is still in progress, and the application can perform other processing. Eventually, the application checks the result of the overlapped request by calling [FT\_W32\_GetOverlappedResult](#_bookmark73). The events that occurred and resulted in this function returning are stored in *lpdwEvent*.

### Examples

1. This example shows how to write 128 bytes to the device using non-overlapped I/O.

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile for non-overlapped i/o FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

DWORD dwEvents;

if (FT\_W32\_WaitCommEvent(ftHandle, &dwEvents, NULL))

; // FT\_W32\_WaitCommEvents OK

else

; // FT\_W32\_WaitCommEvents failed

1. This example shows how to write 128 bytes to the device using overlapped I/O.

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile for overlapped i/o FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

DWORD dwEvents;

DWORD dwRes;

OVERLAPPED osWait = { 0 };

if (!FT\_W32\_WaitCommEvent(ftHandle, &dwEvents, &osWait)) {

if (FT\_W32\_GetLastError(ftHandle) == ERROR\_IO\_PENDING) {

// wait is delayed so do some other stuff until ...

if (!FT\_W32\_GetOverlappedResult(ftHandle, &osWait, &dwRes, FALSE))

; // error

else

}

}

else {

; // FT\_W32\_WaitCommEvent OK

// Events that occurred are stored in dwEvents

// FT\_W32\_WaitCommEvent OK

// Events that occurred are stored in dwEvents

}

## FT\_W32\_PurgeComm

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

This function purges the device.

### Definition

BOOL **FT\_W32\_PurgeComm** (FT\_HANDLE *ftHandle*, DWORD *dwFlags*)

### Parameters

*ftHandle* Handle of the device.

*dwFlags* Specifies the action to take. The action can be a combination of the following:

PURGE\_TXABORT – Terminate outstanding overlapped writes

PURGE\_RXABORT – Terminate outstanding overlapped reads

PURGE\_TXCLEAR – Clear the transmit buffer PURGE\_RXCLEAR – Clear the receive buffer

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Example**

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (FT\_W32\_PurgeComm(ftHandle,PURGE\_TXCLEAR|PURGE\_RXCLEAR))

; // FT\_W32\_PurgeComm OK

else

; // FT\_W32\_PurgeComm failed

## FT\_W32\_GetLastError

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Gets the last error that occurred on the device.

### Definition

DWORD **FT\_W32\_GetLastError** (FT\_HANDLE *ftHandle*)

### Parameters

*ftHandle* Handle of the device.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero. **Remarks**

This function is normally used with overlapped I/O and so is not supported in Windows CE. For a

description of its use, see [FT\_W32\_ReadFile](#_bookmark71) and [FT\_W32\_WriteFile](#_bookmark72).

In Linux and Mac OS X, this function returns a DWORD that directly maps to the FT Errors (for example the FT\_INVALID\_HANDLE error number).

## FT\_W32\_ClearCommError

### Supported Operating Systems

Linux

Mac OS X (10.4 and later) Windows (2000 and later) Windows CE (4.2 and later) **Summary**

Gets information about a communications error and get current status of the device.

### Definition

BOOL **FT\_W32\_ClearCommError** (FT\_HANDLE *ftHandle*, LPDWORD *lpdwErrors,*

LPFTCOMSTAT *lpftComstat*)

### Parameters

*ftHandle* Handle of the device.

*lpdwErrors* Variable that contains the error mask.

*lpftComstat* Pointer to FTCOMSTAT structure.

### Return Value

If the function is successful, the return value is nonzero. If the function is unsuccessful, the return value is zero.

### Example

static COMSTAT oldCS = {0}; static DWORD dwOldErrors = 0;

FT\_HANDLE ftHandle; // setup by FT\_W32\_CreateFile

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); COMSTAT newCS;

DWORD dwErrors;

BOOL bChanged = FALSE;

if (!FT\_W32\_ClearCommError(ftHandle, &dwErrors, (FTCOMSTAT \*)&newCS))

; // FT\_W32\_ClearCommError failed

if (dwErrors != dwOldErrors) { bChanged = TRUE; dwOldErrors = dwErrors;

}

if (memcmp(&oldCS, &newCS, sizeof(FTCOMSTAT))) { bChanged = TRUE;

oldCS = newCS;

}

if (bChanged) {

if (dwErrors & CE\_BREAK)

; // BREAK condition detected if (dwErrors & CE\_FRAME)

; // Framing error detected if (dwErrors & CE\_RXOVER)

; // Receive buffer has overflowed if (dwErrors & CE\_TXFULL)

; // Transmit buffer full if (dwErrors & CE\_OVERRUN)

; // Character buffer overrun if (dwErrors & CE\_RXPARITY)

; // Parity error detected if (newCS.fCtsHold)

; // Transmitter waiting for CTS if (newCS.fDsrHold)

; // Transmitter is waiting for DSR if (newCS.fRlsdHold)

; // Transmitter is waiting for RLSD if (newCS.fXoffHold)

; // Transmitter is waiting because XOFF was received if (newCS.fXoffSent)

; //

if (newCS.fEof)

; // End of file character has been received if (newCS.fTxim)

; // Tx immediate character queued for transmission

// newCS.cbInQue contains number of bytes in receive queue

// newCS.cbOutQue contains number of bytes in transmit queue

}

# Contact Information

|  |  |  |  |
| --- | --- | --- | --- |
| Head Office – Glasgow, UK |  | Branch Office – Tigard, Oregon, USA | |
| Future Technology Devices International Limited Unit 1, 2 Seaward Place, Centurion Business Park Glasgow G41 1HH  United Kingdom  Tel: +44 (0) 141 429 2777  Fax: +44 (0) 141 429 2758 | | Future Technology Devices International Limited (USA) 7130 SW Fir Loop  Tigard, OR 97223-8160 USA  Tel: +1 (503) 547 0988  Fax: +1 (503) 547 0987 | |
| E-mail (Sales) | [sales1@ftdichip.com](mailto:sales1@ftdichip.com) | E-mail (Sales) | [us.sales@ftdichip.com](mailto:us.sales@ftdichip.com) |
| E-mail (Support) | [support1@ftdichip.com](mailto:support1@ftdichip.com) | E-mail (Support) | [us.support@ftdichip.com](mailto:us.support@ftdichip.com) |
| E-mail (General Enquiries) | [admin1@ftdichip.com](mailto:admin1@ftdichip.com) | E-mail (General Enquiries) | [us.admin@ftdichip.com](mailto:us.admin@ftdichip.com) |

|  |  |  |  |
| --- | --- | --- | --- |
| Branch Office – Taipei, Taiwan | | Branch Office – Shanghai, China | |
| Future Technology Devices International Limited (Taiwan)  2F, No. 516, Sec. 1, NeiHu Road Taipei 114  Taiwan, R.O.C.  Tel: +886 (0) 2 8797 1330  Fax: +886 (0) 2 8791 3576 | | Future Technology Devices International Limited (China) Room 1103, No. 666 West Huaihai Road,  Shanghai, 200052 China  Tel: +86 21 62351596  Fax: +86 21 62351595 | |
| E-mail (Sales) | [tw.sales1@ftdichip.com](mailto:tw.sales1@ftdichip.com) | E-mail (Sales) | [cn.sales@ftdichip.com](mailto:cn.sales@ftdichip.com) |
| E-mail (Support) | [tw.support1@ftdichip.com](mailto:tw.support1@ftdichip.com) | E-mail (Support) | [cn.support@ftdichip.com](mailto:cn.support@ftdichip.com) |
| E-mail (General Enquiries) | [tw.admin1@ftdichip.com](mailto:tw.admin1@ftdichip.com) | E-mail (General Enquiries) | [cn.admin@ftdichip.com](mailto:cn.admin@ftdichip.com) |

#### Web Site

[http://ftdichip.com](http://ftdichip.com/)

#### Distributor and Sales Representatives

Please visit the Sales Network page of the [FTDI Web site](http://www.ftdichip.com/) for the contact details of our distributor(s) and sales representative(s) in your country.

System and equipment manufacturers and designers are responsible to ensure that their systems, and any Future Technology Devices International Ltd (FTDI) devices incorporated in their systems, meet all applicable safety, regulatory and system-level performance requirements. All application-related information in this document (including application descriptions, suggested FTDI devices and other materials) is provided for reference only. While FTDI has taken care to assure it is accurate, this information is subject to customer confirmation, and FTDI disclaims all liability for system designs and for any applications assistance provided by FTDI. Use of FTDI devices in life support and/or safety applications is entirely at the user’s risk, and the user agrees to defend, indemnify and hold harmless FTDI from any and all damages, claims, suits or expense resulting from such use. This document is subject to change without notice. No freedom to use patents or other intellectual property rights is implied by the publication of this document. Neither the whole nor any part of the information contained in, or the product described in this document, may be adapted or reproduced in any material or electronic form without the prior written consent of the copyright holder. Future Technology Devices International Ltd, Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom. Scotland Registered Company Number: SC136640

# Appendix A - Type Definitions

UCHAR Unsigned char (1 byte) PUCHAR Pointer to unsigned char PCHAR Pointer to char

DWORD Unsigned long (4 bytes) LPDWORD Pointer to unsigned long FT\_HANDLE Pointer to handle

FT\_STATUS (DWORD)

FT\_OK = 0

FT\_INVALID\_HANDLE = 1

FT\_DEVICE\_NOT\_FOUND = 2

FT\_DEVICE\_NOT\_OPENED = 3

FT\_IO\_ERROR = 4

FT\_INSUFFICIENT\_RESOURCES = 5

FT\_INVALID\_PARAMETER = 6

FT\_INVALID\_BAUD\_RATE = 7

FT\_DEVICE\_NOT\_OPENED\_FOR\_ERASE = 8

FT\_DEVICE\_NOT\_OPENED\_FOR\_WRITE = 9

FT\_FAILED\_TO\_WRITE\_DEVICE = 10

FT\_EEPROM\_READ\_FAILED = 11

FT\_EEPROM\_WRITE\_FAILED = 12

FT\_EEPROM\_ERASE\_FAILED = 13

FT\_EEPROM\_NOT\_PRESENT = 14

FT\_EEPROM\_NOT\_PROGRAMMED = 15

FT\_INVALID\_ARGS = 16

FT\_NOT\_SUPPORTED = 17

FT\_OTHER\_ERROR = 18

Flags (see [FT\_ListDevices](#_bookmark9)) FT\_LIST\_NUMBER\_ONLY = 0x80000000 FT\_LIST\_BY\_INDEX = 0x40000000 FT\_LIST\_ALL = 0x20000000

Flags (see [FT\_OpenEx](#_bookmark11))

FT\_OPEN\_BY\_SERIAL\_NUMBER = 1

FT\_OPEN\_BY\_DESCRIPTION = 2

FT\_OPEN\_BY\_LOCATION = 4

FT\_DEVICE (DWORD)

FT\_DEVICE\_232BM = 0

FT\_DEVICE\_232AM = 1

FT\_DEVICE\_100AX = 2

FT\_DEVICE\_UNKNOWN = 3

FT\_DEVICE\_2232C = 4

FT\_DEVICE\_232R = 5

FT\_DEVICE\_2232H = 6

FT\_DEVICE\_4232H = 7

FT\_DEVICE\_232H = 8

FT\_DEVICE\_X\_SERIES = 9

Driver types FT\_DRIVER\_TYPE\_D2XX 0

FT\_DRIVER\_TYPE\_VCP 1

Word Length (see [FT\_SetDataCharacteristics](#_bookmark17)) FT\_BITS\_8 = 8

FT\_BITS\_7 = 7

Stop Bits ([see FT\_SetDataCharacteristics](#_bookmark17)) FT\_STOP\_BITS\_1 = 0

FT\_STOP\_BITS\_2 = 2

Parity (see [FT\_SetDataCharacteristics](#_bookmark17)) FT\_PARITY\_NONE = 0

FT\_PARITY\_ODD = 1

FT\_PARITY\_EVEN = 2

FT\_PARITY\_MARK = 3

FT\_PARITY\_SPACE = 4

Flow Control (see [FT\_SetFlowControl](#_bookmark19)) FT\_FLOW\_NONE = 0x0000 FT\_FLOW\_RTS\_CTS = 0x0100 FT\_FLOW\_DTR\_DSR = 0x0200 FT\_FLOW\_XON\_XOFF = 0x0400

Purge RX and TX Buffers (see [FT\_Purge](#_bookmark36)) FT\_PURGE\_RX = 1

FT\_PURGE\_TX = 2

Notification Events (see [FT\_SetEventNotification](#_bookmark32)) FT\_EVENT\_RXCHAR = 1

FT\_EVENT\_MODEM\_STATUS = 2

FT\_EVENT\_LINE\_STATUS = 4

Modem Status (see

[FT\_GetModemStatus](#_bookmark25))

CTS = 0x10 DSR = 0x20 RI = 0x40 DCD = 0x80

Line Status (see

[FT\_GetModemStatus](#_bookmark25))

OE = 0x02 PE = 0x04 FE = 0x08 BI = 0x10

Bit Modes (see [FT\_SetBitMode](#_bookmark65)) FT\_BITMODE\_RESET = 0x00 FT\_BITMODE\_ASYNC\_BITBANG = 0x01 FT\_BITMODE\_MPSSE = 0x02

FT\_BITMODE\_SYNC\_BITBANG = 0x04 FT\_BITMODE\_MCU\_HOST = 0x08 FT\_BITMODE\_FAST\_SERIAL = 0x10 FT\_BITMODE\_CBUS\_BITBANG = 0x20 FT\_BITMODE\_SYNC\_FIFO = 0x40

FT232R CBUS EEPROM OPTIONS - Ignored for FT245R (see [FT\_EE\_Program](#_bookmark55) and [FT\_EE\_Read](#_bookmark53)) FT\_232R\_CBUS\_TXDEN = 0x00

FT\_232R\_CBUS\_PWRON = 0x01 FT\_232R\_CBUS\_RXLED = 0x02 FT\_232R\_CBUS\_TXLED = 0x03 FT\_232R\_CBUS\_TXRXLED = 0x04 FT\_232R\_CBUS\_SLEEP = 0x05 FT\_232R\_CBUS\_CLK48 = 0x06 FT\_232R\_CBUS\_CLK24 = 0x07 FT\_232R\_CBUS\_CLK12 = 0x08 FT\_232R\_CBUS\_CLK6 = 0x09 FT\_232R\_CBUS\_IOMODE = 0x0A FT\_232R\_CBUS\_BITBANG\_WR = 0x0B FT\_232R\_CBUS\_BITBANG\_RD = 0x0C

FT232H CBUS EEPROM OPTIONS (see [FT\_EE\_Program](#_bookmark55) and [FT\_EE\_Read](#_bookmark53)) FT\_232H\_CBUS\_TRISTATE = 0x00

FT\_232H\_CBUS\_RXLED = 0x01 FT\_232H\_CBUS\_TXLED = 0x02 FT\_232H\_CBUS\_TXRXLED = 0x03 FT\_232H\_CBUS\_PWREN = 0x04 FT\_232H\_CBUS\_SLEEP = 0x05 FT\_232H\_CBUS\_DRIVE\_0 = 0x06 FT\_232H\_CBUS\_DRIVE\_1 = 0x07 FT\_232H\_CBUS\_IOMODE = 0x08 FT\_232H\_CBUS\_TXDEN = 0x09 FT\_232H\_CBUS\_CLK30 = 0x0A FT\_232H\_CBUS\_CLK15 = 0x0B FT\_232H\_CBUS\_CLK7\_5 = 0x0C

FT X Series CBUS Options EEPROM values (see FT\_EEPROM\_Read and FT\_EEPROM\_Program) FT\_X\_SERIES\_CBUS\_TRISTATE = 0x00

FT\_X\_SERIES\_CBUS\_RXLED = 0x01 FT\_X\_SERIES\_CBUS\_TXLED = 0x02 FT\_X\_SERIES\_CBUS\_TXRXLED = 0x03 FT\_X\_SERIES\_CBUS\_PWREN = 0x04 FT\_X\_SERIES\_CBUS\_SLEEP = 0x05 FT\_X\_SERIES\_CBUS\_DRIVE\_0 = 0x06 FT\_X\_SERIES\_CBUS\_DRIVE\_1 = 0x07 FT\_X\_SERIES\_CBUS\_IOMODE = 0x08 FT\_X\_SERIES\_CBUS\_TXDEN = 0x09 FT\_X\_SERIES\_CBUS\_CLK24 = 0x0A FT\_X\_SERIES\_CBUS\_CLK12 = 0x0B FT\_X\_SERIES\_CBUS\_CLK6 = 0x0C

FT\_X\_SERIES\_CBUS\_BCD\_CHARGER = 0x0D FT\_X\_SERIES\_CBUS\_BCD\_CHARGER\_N = 0x0E FT\_X\_SERIES\_CBUS\_I2C\_TXE = 0x0F FT\_X\_SERIES\_CBUS\_I2C\_RXF = 0x10 FT\_X\_SERIES\_CBUS\_VBUS\_SENSE = 0x11 FT\_X\_SERIES\_CBUS\_BITBANG\_WR = 0x12 FT\_X\_SERIES\_CBUS\_BITBANG\_RD = 0x13 FT\_X\_SERIES\_CBUS\_TIMESTAMP = 0x14 FT\_X\_SERIES\_CBUS\_KEEP\_AWAKE = 0x15

FT\_DEVICE\_LIST\_INFO\_NODE (see [FT\_GetDeviceInfoList](#_bookmark7) and [FT\_GetDeviceInfoDetail](#_bookmark8)) typedef struct \_ft\_device\_list\_info\_node {

DWORD Flags;

DWORD Type;

DWORD ID;

DWORD LocId;

char SerialNumber[16]; char Description[64]; FT\_HANDLE ftHandle;

} FT\_DEVICE\_LIST\_INFO\_NODE;

FT\_FLAGS (see [FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102)) FT\_FLAGS\_OPENED = 0x00000001

FT\_PROGRAM\_DATA\_STRUCTURE

typedef struct ft\_program\_data {

DWORD Signature1; // Header - must be 0x0000000 DWORD Signature2; // Header - must be 0xffffffff

DWORD Version; // Header - FT\_PROGRAM\_DATA version

// 0 = original (FT232B)

// 1 = FT2232 extensions

// 2 = FT232R extensions

// 3 = FT2232H extensions

// 4 = FT4232H extensions

// 5 = FT232H extensions

WORD VendorId; // 0x0403

WORD ProductId; // 0x6001

char \*Manufacturer; // "FTDI"

char \*ManufacturerId; // "FT"

char \*Description; // "USB HS Serial Converter" char \*SerialNumber; // "FT000001" if fixed, or NULL WORD MaxPower; // 0 < MaxPower <= 500

WORD PnP; // 0 = disabled, 1 = enabled

WORD SelfPowered; // 0 = bus powered, 1 = self powered WORD RemoteWakeup; // 0 = not capable, 1 = capable

//

// Rev4 (FT232B) extensions

//

UCHAR Rev4; // non-zero if Rev4 chip, zero otherwise

UCHAR IsoIn; // non-zero if in endpoint is isochronous

UCHAR IsoOut; // non-zero if out endpoint is isochronous UCHAR PullDownEnable; // non-zero if pull down enabled

UCHAR SerNumEnable; // non-zero if serial number to be used

UCHAR USBVersionEnable; // non-zero if chip uses USBVersion WORD USBVersion; // BCD (0x0200 => USB2)

//

// Rev 5 (FT2232) extensions

//

UCHAR Rev5; // non-zero if Rev5 chip, zero otherwise

UCHAR IsoInA; // non-zero if in endpoint is isochronous

UCHAR IsoInB; // non-zero if in endpoint is isochronous

UCHAR IsoOutA; // non-zero if out endpoint is isochronous

UCHAR IsoOutB; // non-zero if out endpoint is isochronous UCHAR PullDownEnable5; // non-zero if pull down enabled

UCHAR SerNumEnable5; // non-zero if serial number to be used UCHAR USBVersionEnable5; // non-zero if chip uses USBVersion WORD USBVersion5; // BCD (0x0200 => USB2)

UCHAR AIsHighCurrent; // non-zero if interface is high current UCHAR BIsHighCurrent; // non-zero if interface is high current UCHAR IFAIsFifo; // non-zero if interface is 245 FIFO

UCHAR IFAIsFifoTar; // non-zero if interface is 245 FIFO CPU target UCHAR IFAIsFastSer; // non-zero if interface is Fast serial

UCHAR AIsVCP; // non-zero if interface is to use VCP drivers

UCHAR IFBIsFifo; // non-zero if interface is 245 FIFO

UCHAR IFBIsFifoTar; // non-zero if interface is 245 FIFO CPU target UCHAR IFBIsFastSer; // non-zero if interface is Fast serial

UCHAR BIsVCP; // non-zero if interface is to use VCP drivers

//

// Rev 6 (FT232R) extensions

//

UCHAR UseExtOsc; // Use External Oscillator UCHAR HighDriveIOs; // High Drive I/Os UCHAR EndpointSize; // Endpoint size

UCHAR PullDownEnableR; // non-zero if pull down enabled UCHAR SerNumEnableR; // non-zero if serial number to be used UCHAR InvertTXD; // non-zero if invert TXD

UCHAR InvertRXD; // non-zero if invert RXD

UCHAR InvertRTS; // non-zero if invert RTS

UCHAR InvertCTS; // non-zero if invert CTS

UCHAR InvertDTR; // non-zero if invert DTR

UCHAR InvertDSR; // non-zero if invert DSR

UCHAR InvertDCD; // non-zero if invert DCD

UCHAR InvertRI; // non-zero if invert RI

UCHAR Cbus0; // Cbus Mux control

UCHAR Cbus1; // Cbus Mux control

UCHAR Cbus2; // Cbus Mux control

UCHAR Cbus3; // Cbus Mux control

UCHAR Cbus4; // Cbus Mux control

UCHAR RIsD2XX; // non-zero if using D2XX driver

//

// Rev 7 (FT2232H) Extensions

//

UCHAR PullDownEnable7; // non-zero if pull down enabled UCHAR SerNumEnable7; // non-zero if serial number to be used UCHAR ALSlowSlew; // non-zero if AL pins have slow slew UCHAR ALSchmittInput; // non-zero if AL pins are Schmitt input

UCHAR ALDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR AHSlowSlew; // non-zero if AH pins have slow slew UCHAR AHSchmittInput; // non-zero if AH pins are Schmitt input UCHAR AHDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR BLSlowSlew; // non-zero if BL pins have slow slew UCHAR BLSchmittInput; // non-zero if BL pins are Schmitt input UCHAR BLDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA

designs

UCHAR BHSlowSlew; // non-zero if BH pins have slow slew UCHAR BHSchmittInput; // non-zero if BH pins are Schmitt input UCHAR BHDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR IFAIsFifo7; // non-zero if interface is 245 FIFO

UCHAR IFAIsFifoTar7; // non-zero if interface is 245 FIFO CPU target UCHAR IFAIsFastSer7; // non-zero if interface is Fast serial

UCHAR AIsVCP7; // non-zero if interface is to use VCP drivers UCHAR IFBIsFifo7; // non-zero if interface is 245 FIFO

UCHAR IFBIsFifoTar7; // non-zero if interface is 245 FIFO CPU target UCHAR IFBIsFastSer7; // non-zero if interface is Fast serial

UCHAR BIsVCP7; // non-zero if interface is to use VCP drivers

UCHAR PowerSaveEnable; // non-zero if using BCBUS7 to save power for self-powered

//

// Rev 8 (FT4232H) Extensions

//

UCHAR PullDownEnable8; // non-zero if pull down enabled UCHAR SerNumEnable8; // non-zero if serial number to be used UCHAR ASlowSlew; // non-zero if AL pins have slow slew UCHAR ASchmittInput; // non-zero if AL pins are Schmitt input

UCHAR ADriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR BSlowSlew; // non-zero if AH pins have slow slew UCHAR BSchmittInput; // non-zero if AH pins are Schmitt input UCHAR BDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR CSlowSlew; // non-zero if BL pins have slow slew UCHAR CSchmittInput; // non-zero if BL pins are Schmitt input UCHAR CDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR DSlowSlew; // non-zero if BH pins have slow slew UCHAR DSchmittInput; // non-zero if BH pins are Schmitt input UCHAR DDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA

UCHAR ARIIsTXDEN; // non-zero if port A uses RI as RS485 TXDEN UCHAR BRIIsTXDEN; // non-zero if port B uses RI as RS485 TXDEN UCHAR CRIIsTXDEN; // non-zero if port C uses RI as RS485 TXDEN UCHAR DRIIsTXDEN; // non-zero if port D uses RI as RS485 TXDEN UCHAR AIsVCP8; // non-zero if interface is to use VCP drivers UCHAR BIsVCP8; // non-zero if interface is to use VCP drivers UCHAR CIsVCP8; // non-zero if interface is to use VCP drivers UCHAR DIsVCP8; // non-zero if interface is to use VCP drivers

//

// Rev 9 (FT232H) Extensions

//

UCHAR PullDownEnableH; // non-zero if pull down enabled UCHAR SerNumEnableH; // non-zero if serial number to be used UCHAR ACSlowSlewH; // non-zero if AC pins have slow slew UCHAR ACSchmittInputH; // non-zero if AC pins are Schmitt input

UCHAR ACDriveCurrentH; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR ADSlowSlewH; // non-zero if AD pins have slow slew UCHAR ADSchmittInputH; // non-zero if AD pins are Schmitt input UCHAR ADDriveCurrentH; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR Cbus0H; // Cbus Mux control

UCHAR Cbus1H; // Cbus Mux control

UCHAR Cbus2H; // Cbus Mux control

UCHAR Cbus3H; // Cbus Mux control

UCHAR Cbus4H; // Cbus Mux control

UCHAR Cbus5H; // Cbus Mux control

UCHAR Cbus6H; // Cbus Mux control

UCHAR Cbus7H; // Cbus Mux control

UCHAR Cbus8H; // Cbus Mux control

UCHAR Cbus9H; // Cbus Mux control

UCHAR IsFifoH; // non-zero if interface is 245 FIFO

designs

UCHAR IsFifoTarH; // non-zero if interface is 245 FIFO CPU target UCHAR IsFastSerH; // non-zero if interface is Fast serial

UCHAR IsFT1248H; // non-zero if interface is FT1248

UCHAR FT1248CpolH; // FT1248 clock polarity - clock idle high (1) or clock idle low (0) UCHAR FT1248LsbH; // FT1248 data is LSB (1) or MSB (0)

UCHAR FT1248FlowControlH; // FT1248 flow control enable

UCHAR IsVCPH; // non-zero if interface is to use VCP drivers

UCHAR PowerSaveEnableH; // non-zero if using ACBUS7 to save power for self-powered

} FT\_PROGRAM\_DATA, \*PFT\_PROGRAM\_DATA;

EEPROM\_HEADER STRUCTURE (See FT\_EEPROM\_Read and FT\_EEPROM\_Program) typedef struct ft\_eeprom\_header {

FT\_DEVICE deviceType; // FTxxxx device type to be programmed

// Device descriptor options

WORD VendorId; // 0x0403

WORD ProductId; // 0x6001

UCHAR SerNumEnable; // non-zero if serial number to be used

// Config descriptor options

WORD MaxPower; // 0 < MaxPower <= 500

UCHAR SelfPowered; // 0 = bus powered, 1 = self powered

UCHAR RemoteWakeup; // 0 = not capable, 1 = capable

// Hardware options

UCHAR PullDownEnable; // non-zero if pull down in suspend enabled

} FT\_EEPROM\_HEADER, \*PFT\_EEPROM\_HEADER;

FT232B EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_232b {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

} FT\_EEPROM\_232B, \*PFT\_EEPROM\_232B;

FT2232 EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_2232 {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

// Drive options

UCHAR AIsHighCurrent; // non-zero if interface is high current

UCHAR BIsHighCurrent; // non-zero if interface is high current

// Hardware options

UCHAR AIsFifo; // non-zero if interface is 245 FIFO

UCHAR AIsFifoTar; // non-zero if interface is 245 FIFO CPU target

UCHAR AIsFastSer; // non-zero if interface is Fast serial

UCHAR BIsFifo; // non-zero if interface is 245 FIFO

UCHAR BIsFifoTar; // non-zero if interface is 245 FIFO CPU target

UCHAR BIsFastSer; // non-zero if interface is Fast serial

// Driver option

UCHAR ADriverType; //

UCHAR BDriverType; //

} FT\_EEPROM\_2232, \*PFT\_EEPROM\_2232;

FT232R EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_232r {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

// Drive options

UCHAR IsHighCurrent; // non-zero if interface is high current

// Hardware options

UCHAR UseExtOsc; // Use External Oscillator

UCHAR InvertTXD; // non-zero if invert TXD

UCHAR InvertRXD; // non-zero if invert RXD

UCHAR InvertRTS; // non-zero if invert RTS

UCHAR InvertCTS; // non-zero if invert CTS

UCHAR InvertDTR; // non-zero if invert DTR

UCHAR InvertDSR; // non-zero if invert DSR

UCHAR InvertDCD; // non-zero if invert DCD UCHAR InvertRI; // non-zero if invert RI UCHAR Cbus0; // Cbus Mux control

UCHAR Cbus1; // Cbus Mux control

UCHAR Cbus2; // Cbus Mux control

UCHAR Cbus3; // Cbus Mux control

UCHAR Cbus4; // Cbus Mux control

// Driver option

UCHAR DriverType; //

} FT\_EEPROM\_232R, \*PFT\_EEPROM\_232R;

FT2232H EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_2232h {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

// Drive options

UCHAR ALSlowSlew; // non-zero if AL pins have slow slew UCHAR ALSchmittInput; // non-zero if AL pins are Schmitt input UCHAR ALDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR AHSlowSlew; // non-zero if AH pins have slow slew UCHAR AHSchmittInput; // non-zero if AH pins are Schmitt input UCHAR AHDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR BLSlowSlew; // non-zero if BL pins have slow slew UCHAR BLSchmittInput; // non-zero if BL pins are Schmitt input UCHAR BLDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR BHSlowSlew; // non-zero if BH pins have slow slew UCHAR BHSchmittInput; // non-zero if BH pins are Schmitt input UCHAR BHDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA

// Hardware options

UCHAR AIsFifo; // non-zero if interface is 245 FIFO

UCHAR AIsFifoTar; // non-zero if interface is 245 FIFO CPU target

UCHAR AIsFastSer; // non-zero if interface is Fast serial

UCHAR BIsFifo; // non-zero if interface is 245 FIFO

UCHAR BIsFifoTar; // non-zero if interface is 245 FIFO CPU target

UCHAR BIsFastSer; // non-zero if interface is Fast serial

UCHAR PowerSaveEnable; // non-zero if using BCBUS7 to save power for

// self-powered designs

// Driver option

UCHAR ADriverType; //

UCHAR BDriverType; //

} FT\_EEPROM\_2232H, \*PFT\_EEPROM\_2232H;

FT4232H EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_4232h {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

// Drive options

UCHAR ASlowSlew; // non-zero if A pins have slow slew UCHAR ASchmittInput; // non-zero if A pins are Schmitt input

UCHAR ADriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR BSlowSlew; // non-zero if B pins have slow slew

UCHAR BSchmittInput; // non-zero if B pins are Schmitt input UCHAR BDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR CSlowSlew; // non-zero if C pins have slow slew

UCHAR CSchmittInput; // non-zero if C pins are Schmitt input UCHAR CDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR DSlowSlew; // non-zero if D pins have slow slew

UCHAR DSchmittInput; // non-zero if D pins are Schmitt input UCHAR DDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA

// Hardware options

UCHAR ARIIsTXDEN; // non-zero if port A uses RI as RS485 TXDEN UCHAR BRIIsTXDEN; // non-zero if port B uses RI as RS485 TXDEN UCHAR CRIIsTXDEN; // non-zero if port C uses RI as RS485 TXDEN UCHAR DRIIsTXDEN; // non-zero if port D uses RI as RS485 TXDEN

// Driver option

UCHAR ADriverType; //

UCHAR BDriverType; //

UCHAR CDriverType; //

UCHAR DDriverType; //

} FT\_EEPROM\_4232H, \*PFT\_EEPROM\_4232H;

// FT232H EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_232h {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

// Drive options

UCHAR ACSlowSlew; // non-zero if AC bus pins have slow slew UCHAR ACSchmittInput; // non-zero if AC bus pins are Schmitt input UCHAR ACDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR ADSlowSlew; // non-zero if AD bus pins have slow slew UCHAR ADSchmittInput; // non-zero if AD bus pins are Schmitt input UCHAR ADDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA

// CBUS options

UCHAR Cbus0; // Cbus Mux control

UCHAR Cbus1; // Cbus Mux control

UCHAR Cbus2; // Cbus Mux control

UCHAR Cbus3; // Cbus Mux control

UCHAR Cbus4; // Cbus Mux control

UCHAR Cbus5; // Cbus Mux control

low (0)

UCHAR Cbus6; // Cbus Mux control

UCHAR Cbus7; // Cbus Mux control

UCHAR Cbus8; // Cbus Mux control

UCHAR Cbus9; // Cbus Mux control

// FT1248 options

UCHAR FT1248Cpol; // FT1248 clock polarity - clock idle high (1) or clock idle

UCHAR FT1248Lsb; // FT1248 data is LSB (1) or MSB (0)

UCHAR FT1248FlowControl; // FT1248 flow control enable

// Hardware options

UCHAR IsFifo; // non-zero if interface is 245 FIFO

UCHAR IsFifoTar; // non-zero if interface is 245 FIFO CPU target UCHAR IsFastSer; // non-zero if interface is Fast serial

UCHAR IsFT1248 // non-zero if interface is FT1248 UCHAR PowerSaveEnable;

// Driver option UCHAR DriverType;

} FT\_EEPROM\_232H, \*PFT\_EEPROM\_232H;

FT X Series EEPROM structure for use with FT\_EEPROM\_Read and FT\_EEPROM\_Program typedef struct ft\_eeprom\_x\_series {

// Common header

FT\_EEPROM\_HEADER common;// common elements for all device EEPROMs

// Drive options

UCHAR ACSlowSlew; // non-zero if AC bus pins have slow slew UCHAR ACSchmittInput; // non-zero if AC bus pins are Schmitt input UCHAR ACDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA UCHAR ADSlowSlew; // non-zero if AD bus pins have slow slew UCHAR ADSchmittInput; // non-zero if AD bus pins are Schmitt input UCHAR ADDriveCurrent; // valid values are 4mA, 8mA, 12mA, 16mA

// CBUS options

UCHAR Cbus0; // Cbus Mux control

UCHAR Cbus1; // Cbus Mux control

UCHAR Cbus2; // Cbus Mux control

UCHAR Cbus3; // Cbus Mux control

UCHAR Cbus4; // Cbus Mux control

UCHAR Cbus5; // Cbus Mux control

UCHAR Cbus6; // Cbus Mux control

// UART signal options

UCHAR InvertTXD; // non-zero if invert TXD

UCHAR InvertRXD; // non-zero if invert RXD

UCHAR InvertRTS; // non-zero if invert RTS

UCHAR InvertCTS; // non-zero if invert CTS

UCHAR InvertDTR; // non-zero if invert DTR

UCHAR InvertDSR; // non-zero if invert DSR

UCHAR InvertDCD; // non-zero if invert DCD

UCHAR InvertRI; // non-zero if invert RI

// Battery Charge Detect options

UCHAR BCDEnable; // Enable Battery Charger Detection

UCHAR BCDForceCbusPWREN; // asserts the power enable signal on CBUS when charging port detected

UCHAR BCDDisableSleep; // forces the device never to go into sleep mode

// I2C options

WORD I2CSlaveAddress; // I2C slave device address

DWORD I2CDeviceId; // I2C device ID

UCHAR I2CDisableSchmitt; // Disable I2C Schmitt trigger

// FT1248 options

UCHAR FT1248Cpol; // FT1248 clock polarity - clock idle high (1) or clock idle low (0)

UCHAR FT1248Lsb; // FT1248 data is LSB (1) or MSB (0)

UCHAR FT1248FlowControl; // FT1248 flow control enable

// Hardware options

UCHAR RS485EchoSuppress; UCHAR PowerSaveEnable;

// Driver option UCHAR DriverType;

} FT\_EEPROM\_X\_SERIES, \*PFT\_EEPROM\_X\_SERIES;

Win32

OPEN\_EXISTING = 3 FILE\_ATTRIBUTE\_NORMAL = 0x00000080 FILE\_FLAG\_OVERLAPPED = 0x40000000 GENERIC\_READ = 0x80000000 GENERIC\_WRITE = 0x40000000

OVERLAPPED structure typedef struct \_OVERLAPPED {

ULONG\_PTR Internal; ULONG\_PTR InternalHigh; union {

struct {

};

DWORD Offset; DWORD OffsetHigh;

PVOID Pointer;

};

HANDLE hEvent;

} OVERLAPPED, \*LPOVERLAPPED;

CLRDTR = 6 – Clear the DTR signal CLRRTS = 4 – Clear the RTS signal SETDTR = 5 – Set the DTR signal SETRTS = 3 – Set the RTS signal SETBREAK = 8 – Set the BREAK condition

CLRBREAK = 9 – Clear the BREAK condition

MS\_CTS\_ON = 0x0010 – Clear To Send (CTS) is on MS\_DSR\_ON = 0x0020 – Data Set Ready (DSR) is on MS\_RING\_ON = 0x0040 – Ring Indicator (RI) is on

MS\_RLSD\_ON = 0x0080 – Receive Line Signal Detect (RLSD) is on

FTDCB structure typedef struct \_FTDCB {

DWORD DCBlength; // sizeof(FTDCB)

DWORD BaudRate; // Baud rate at which running DWORD fBinary: 1; // Binary Mode (skip EOF check) DWORD fParity: 1; // Enable parity checking

DWORD fOutxCtsFlow:1; // CTS handshaking on output DWORD fOutxDsrFlow:1; // DSR handshaking on output DWORD fDtrControl:2; // DTR Flow control

DWORD fDsrSensitivity:1; // DSR Sensitivity

DWORD fTXContinueOnXoff: 1; // Continue TX when Xoff sent DWORD fOutX: 1; // Enable output X-ON/X-OFF

DWORD fInX: 1; // Enable input X-ON/X-OFF DWORD fErrorChar: 1; // Enable Err Replacement DWORD fNull: 1; // Enable Null stripping

DWORD fRtsControl:2; // Rts Flow control

DWORD fAbortOnError:1; // Abort all reads and writes on Error DWORD fDummy2:17; // Reserved

WORD wReserved; // Not currently used WORD XonLim; // Transmit X-ON threshold WORD XoffLim; // Transmit X-OFF threshold BYTE ByteSize; // Number of bits/byte, 7-8

BYTE Parity; // 0-4=None,Odd,Even,Mark,Space BYTE StopBits; // 0,2 = 1, 2

char XonChar; // Tx and Rx X-ON character char XoffChar; // Tx and Rx X-OFF character char ErrorChar; // Error replacement char char EofChar; // End of Input character

char EvtChar; // Received Event character WORD wReserved1; // Fill

} FTDCB, \*LPFTDCB;

FTTIMEOUTS structure typedef struct \_FTTIMEOUTS {

DWORD ReadIntervalTimeout; // Maximum time between read chars DWORD ReadTotalTimeoutMultiplier; // Multiplier of characters DWORD ReadTotalTimeoutConstant; // Constant in milliseconds DWORD WriteTotalTimeoutMultiplier; // Multiplier of characters DWORD WriteTotalTimeoutConstant; // Constant in milliseconds

} FTTIMEOUTS, \*LPFTTIMEOUTS;

EV\_BREAK = 0x0040 – BREAK condition detected EV\_CTS = 0x0008 – Change in Clear To Send (CTS) EV\_DSR = 0x0010 – Change in Data Set Ready (DSR) EV\_ERR = 0x0080 – Error in line status

EV\_RING = 0x0100 – Change in Ring Indicator (RI)

EV\_RLSD = 0x0020 – Change in Receive Line Signal Detect (RLSD) EV\_RXCHAR = 0x0001 – Character received

EV\_RXFLAG = 0x0002 – Event character received EV\_TXEMPTY = 0x0004 – Transmitter empty

PURGE\_TXABORT = 0x0001 – Terminate outstanding overlapped writes PURGE\_RXABORT = 0x0002 – Terminate outstanding overlapped reads PURGE\_TXCLEAR = 0x0004 – Clear the transmit buffer PURGE\_RXCLEAR = 0x0008 – Clear the receive buffer

FTCOMSTAT structure typedef struct \_FTCOMSTAT {

DWORD fCtsHold : 1; DWORD fDsrHold : 1; DWORD fRlsdHold : 1; DWORD fXoffHold : 1; DWORD fXoffSent : 1; DWORD fEof : 1;

DWORD fTxim : 1; DWORD fReserved : 25; DWORD cbInQue; DWORD cbOutQue;

} FTCOMSTAT, \*LPFTCOMSTAT;

# Appendix B – References

## Document References

NA

## Acronyms and Abbreviations

|  |  |
| --- | --- |
| Terms | Description |
| CDM | Combined Driver Model. Windows driver package which incorporates both D2XX and VCP drivers. |
| D2XX | FTDI’s proprietary “direct” driver interface via FTD2XX.DLL |
| VCP | Virtual COM Port |

# Appendix C – List of Tables & Figures

## List of Tables

No table of figures entries found.

## List of Figures

[Figure 2.1 Windows CDM Driver Architecture **6**](#_bookmark2)

# Appendix D - Revision History

Document Title: D2XX Programmer's Guide Document Reference No.: FT\_000071

Clearance No.: FTDI# 170

Product Page: <https://ftdichip.com/product-category/products/ic/>

Document Feedback: [Send Feedback](mailto:docufeedback@ftdichip.com)

|  |  |  |
| --- | --- | --- |
| **Revision** | **Changes** | **Date** |
| 1.00 | Initial release in new format  Includes all functions in CDM driver 2.04.06 | August 2008 |
| 1.01 | Includes FT4232H and FT2232H Updated addresses | January 2009 |
| 1.02 | Page 65 – removed FT232R and FT245R reference from MCU host emulation and Fast Opto modes | January, 2010 |
| 1.03 | Corrected section 3.32 (FT\_Purge) Updated Contact details | 2010-09-08 |
| 1.04 | Added 245 Synchronous FIFO mode code in section 5.3 | 2010-10-28 |
| 1.1 | Corrected previous editing errors to the document by re-adding FT4232H and FT2232H extensions | 2010-11-04 |
| 1.2 | Added references to FT232H including EEPROM format  Numerous formatting fixes  Expanded definitions in appendix to reflect updates in CDM 2.08.14 header file. | 2011-04-25 |
| 1.3 | Added sections 4.11 and 4.12 for FT\_EEPROM\_Read and FT\_EEPROM\_Program  Updated ftd2xx.h attachemnet at the end of the doc. | 2012-02-23 |
| 1.4 | Document Template changes  Changes to Appendix A - Type Definitions  Added information on Manufacturer, ManufacturerId, Description and Serial Number | 2019-06-24 |
| 1.5 | All code snippets were updated and tested with Visual Studio 2019 conformance mode enabled.  Added Linux support to FT\_EEPROM\_Read and FT\_EEPROM\_Program.  Minor changes made to FT\_EE\_UASize, FT\_SetBitMode, FT\_GetBitMode and FT\_W32\_WaitCommEvent. | 07-09-2023 |