Future Technology Devices International Ltd.

Software Application Development

D2XX Programmer's Guide

Document Reference No.: FT_000071

Version 1.02

Issue Date: 2010-01-19

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.

Future Technology Devices International Limited (FTDI)

Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
E-Mail (Support): support1@ftdichip.com Web: http://www.ftdichip.com

Table of Contents

1 Preface

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

1.1 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

Table 1.1 Acronyms and Abbreviations

2 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 Architecture illustrates the architecture of the Windows CDM driver.

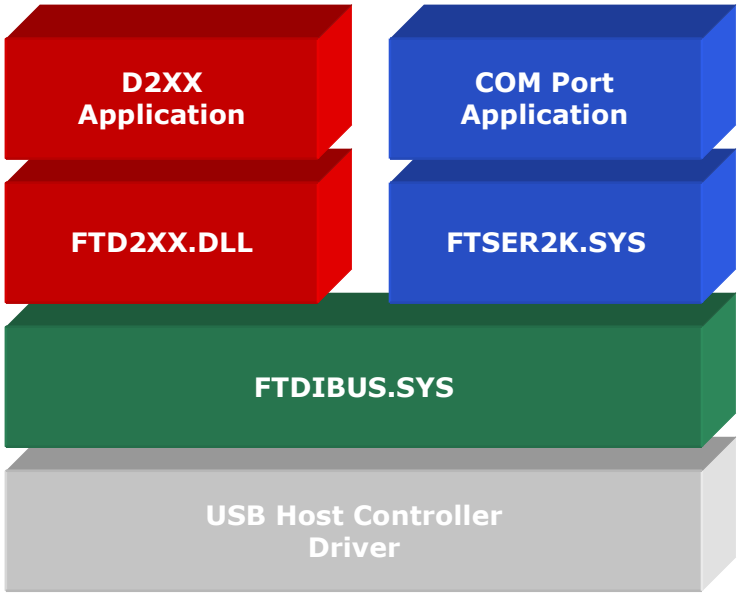

Figure 2.1 Windows CDM Driver Architecture

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.

3 D2XX Classic Functions

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

3.1 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, FT_Open, FT_OpenEx or FT_CreateDeviceInfoList.

3.2 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.

3.3 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 or FT_GetDeviceInfoDetailFT_GetDeviceInfoDetail.

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
}
```

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

Location ID information is not returned for devices that are open when FT_CreateDeviceInfoList 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 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 Appendix A – Type Definitions. 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 contains all available data on each device. The structure of FT_DEVICE_LIST_INFO_NODES is given in the Appendix. The storage for the list must be allocated by the application. The number of devices returned by FT_CreateDeviceInfoList can be used to do this.

When programming in Visual Basic, LabVIEW or similar languages, FT_GetDeviceInfoDetail 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);
		}
	}
}
```

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

The index value is zero-based.

The flag value is a 4-byte bit map containing miscellaneous data as defined Appendix A – Type Definitions. 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 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 structures, use FT_CreateDeviceInfoList.

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);
	}
}
```

3.6 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, FT_GetDeviceInfoList and FT_GetDeviceInfoDetail 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* 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* bit is set in *dwFlags*, the serial number string will be returned; if *FT_OPEN_BY_DESCRIPTION* bit is set in *dwFlags*, the product description string will be returned; if *FT_OPEN_BY_LOCATION* 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* and *FT_OPEN_BY_SERIAL_NUMBER* or *FT_OPEN_BY_DESCRIPTION* 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* is returned for an invalid index.

It can be used to return device string information for all connected devices. If *FT_LIST_ALL* and *FT_OPEN_BY_SERIAL_NUMBER* or *FT_OPEN_BY_DESCRIPTION* 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* and *FT_OPEN_BY_LOCATION* 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* 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* and *FT_OPEN_BY_LOCATION* 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
}
```

2. 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.

3. 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.

4. 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.

3.7 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.

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
}
```

3.8 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 or FT_ListDevices 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, FT_OPEN_BY_DESCRIPTION or FT_OPEN_BY_LOCATION.
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*, *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*, *pvArg1* is interpreted as a pointer to a null-terminated string that represents the device description; if *dwFlags* is *FT_OPEN_BY_LOCATION*, *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("FT000001",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle1);
if (ftStatus == FT_OK) {
	// success - device with serial number "FT000001" is open
}
else {
	// failure
}
```

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

```
ftStatus = FT_OpenEx("USB Serial Converter",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
if (ftStatus == FT_OK) {
	// success - device with device description "USB Serial Converter" is open
}
else {
	// failure
}
```

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

```
ftStatus = FT_OpenEx("FT000001",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle1);
ftStatus2 = FT_OpenEx("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
```

```
}
```

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

```
ftStatus = FT_OpenEx("USB Serial Converter",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
ftStatus2 = FT_OpenEx("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
}
```

5. Open a device at location 23

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle1);
if (ftStatus == FT_OK) {
	// success - device at location 23 is open
}
else {
	// failure
}
```

6. Open 2 devices at locations 23 and 31

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle1);
dwLoc = 0x31;
ftStatus2 = FT_OpenEx(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
}
```

3.9 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
}
```

3.10 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 or FT_GetQueueStatus, 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, 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);
```

2. 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);
```

3.11 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);
```

3.12 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);
```

3.13 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 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.

3.14 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 or FT_BITS_7.
uStopBits	Number of stop bits - must be FT_STOP_BITS_1 or FT_STOP_BITS_2.
uParity	Parity - must be FT_PARITY_NONE, FT_PARITY_ODD, FT_PARITY_EVEN, FT_PARITY_MARK or FT_PARITY_SPACE.

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);
```

3.15 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);
```

3.16 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, FT_FLOW_RTS_CTS, FT_FLOW_DTR_DSR or FT_FLOW_XON_XOFF.
uXon	Character used to signal Xon. Only used if flow control is FT_FLOW_XON_XOFF.
uXoff	Character used to signal Xoff. Only used if flow control is FT_FLOW_XON_XOFF.

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);
```

3.17 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);
```

3.18 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);
```

3.19 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);
```

3.20 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);
```

3.21 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) = 0x10, Data Set Ready (DSR) = 0x20, Ring Indicator (RI) = 0x40, Data Carrier Detect (DCD) = 0x80.

The line status is bit-mapped as follows: Overrun Error (OE) = 0x02, Parity Error (PE) = 0x04, Framing Error (FE) = 0x08, Break Interrupt (BI) = 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);
```

3.22 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);
```

3.23 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_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!)
	// deviceID contains encoded device ID
	// SerialNumber, Description contain 0-terminated strings
}
else {
	// FT_GetDeviceType FAILED!
}
```

```
FT_Close(ftHandle);
```

3.24 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);
}
```

3.25 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");
```

3.26 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; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;
LONG lComPortNumber;

ftStatus = FT_GetComPortNumber(ftHandle,&lComPortNumber);
if (status == FT_OK) {
	if (lComPortNumber == -1) {
		// No COM port assigned
	}
	else {
		// COM port assigned with number held in lComPortNumber
	}
}
else {
	// FT_GetComPortNumber FAILED!
}
FT_Close(ftHandle);
```

3.27 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.

3.28 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* is set in *dwEventMask*, the event will be set when a character has been received by the device.

If *FT_EVENT_MODEM_STATUS* 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* 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; // handle of an open device
FT_STATUS ftStatus;
HANDLE hEvent;
DWORD EventMask;

hEvent = CreateEvent(
                    NULL,
                    false, // auto-reset event
                    false, // non-signalled state
                    ""
                    );
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
}
```

2. 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);
```

3.29 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.

3.30 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; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_SetBreakOn(ftHandle);
if (ftStatus == FT_OK) {
	// FT_SetBreakOn OK
}
else {
	// FT_SetBreakOn failed
}

FT_Close(ftHandle);
```

3.31 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; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_SetBreakOff(ftHandle);
if (ftStatus == FT_OK) {
	// FT_SetBreakOff OK
}
else {
	// FT_SetBreakOff failed
}

FT_Close(ftHandle);
```

3.32 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_SetChars** (FT_HANDLE *ftHandle,* DWORD *dwMask*)

Parameters

ftHandle Handle of the device.

uEventCh Combination of FT_PURGE_RX and FT_PURGE_TX.

Return Value

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

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

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);
```

3.33 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; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_ResetDevice(ftHandle);
if (ftStatus == FT_OK) {
	// FT_ResetDevice OK
}
else {
	// FT_ResetDevice failed
}
FT_Close(ftHandle);
```

3.34 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.

Example

```
FT_HANDLE ftHandle; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_ResetPort(ftHandle);
if (ftStatus == FT_OK) {
	// Port has been reset
}
else {
	// FT_ResetPort FAILED!
}
```

3.35 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; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_CyclePort(ftHandle);
if (ftStatus == FT_OK) {
	// Port has been cycled.
	// Close the handle.
	ftStatus = FT_Close(ftHandle);
}
else {
	// FT_CyclePort FAILED!
}
```

3.36 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;
}
```

3.37 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;
}
```

2. 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;
}
```

3.38 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; // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;
DWORD dwRetryCount;

dwRetryCount = 100;
ftStatus = FT_SetResetPipeRetryCount(ftHandle,dwRetryCount);
if (ftStatus == FT_OK) {
	// ResetPipeRetryCount set to 100
}
else {
	// FT_SetResetPipeRetryCount FAILED!
}
```

3.39 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 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);
```

3.40 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.

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);
```

3.41 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);
```

3.42 FT_IoCtl

Undocumented function.

3.43 FT_SetWaitMask

Undocumented function.

3.44 FT_WaitOnMask

Undocumented function.

4 EEPROM Programming Interface Functions

FTDI device EEPROMs can be both read and programmed using the functions listed in this section.

4.1 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.

4.2 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.

4.3 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.

4.4 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 = 0x00000004;		// EEPROM structure with FT4232H 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);
```

4.5 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 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.

4.6 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
       "FTDI",                          // Manufacturer
       "FT",                      // Manufacturer ID
       "USB HS Serial Converter", // Description
       "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

};
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);
}
```

4.7 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 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:

4.8 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_UASizeWrite** (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);
```

4.9 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.

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);
```

4.10 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.

Example

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus = FT_Open(0, &ftHandle);

if (ftStatus != FT_OK) {
        // FT_Open FAILED!
}
```

```
char *buffer = "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);
```

5 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.

5.1 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);
```

5.2 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);
```

5.3 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 (FT4232H, FT2232H and FT2232 devices only) 0x4 = Synchronous Bit Bang (FT4232H, FT2232H, FT232R, FT245R and FT2232 devices only)

0x8 = MCU Host Bus Emulation Mode (FT4232H, FT2232H and FT2232 devices only)

0x10 = Fast Opto-Isolated Serial Mode (FT4232H, FT2232H and FT2232 devices only)

0x20 = CBUS Bit Bang Mode (FT232R devices only)

0x40 = Single Channel Synchronous 245 FIFO Mode (FT2232H 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);
```

5.4 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);
```

5.5 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);
```

6 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.

6.1 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, FT_GetDeviceInfoDetail or FT_ListDevices 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 if *lpszName* is the device's serial number, and FT_OPEN_BY_DESCRIPTION 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* or *FT_OPEN_BY_DESCRIPTION* 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* 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* or *FT_OPEN_BY_DESCRIPTION* or *FT_OPEN_BY_LOCATION*; 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(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
```

2. 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(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
```

3. 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((PVOID) 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
```

6.2 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(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);
}
```

6.3 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 or FT_GetQueueStatus, 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, 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 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.

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
char Buf[256];
DWORD dwToRead = 256;
DWORD dwRead;

if (FT_W32_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osRead)) {
       if (dwToRead == dwRead){
              // FT_W32_ReadFile OK
       }
       else{
```

```
                // FT_W32_ReadFile timeout
        }
}
else{
        // FT_W32_ReadFile failed
}
```

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

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile for overlapped i/o
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{
                                // FT_W32_ReadFile timeout
                        }
                }
        }
}
else {
        // FT_W32_ReadFile OK
}
CloseHandle (osRead.hEvent);
```

6.4 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, 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 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.

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; // setup by FT_W32_CreateFile for overlapped i/o
char Buf[128]; // contains data to write to the device
DWORD dwToWrite = 128;
DWORD dwWritten;

if (FT_W32_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) {
	if (dwToWrite == dwWritten){
		// FT_W32_WriteFile OK
	}
	else{
		// FT_W32_WriteFile timeout
	}
}
else{
	// FT_W32_WriteFile failed
}
```

2. 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
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
}
```

6.5 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 and FT_W32_WriteFile.

6.6 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,CLRDTS); // Clear the DTR signal
FT_W32_EscapeCommFunction(ftHandle,SETRTS); // Set the RTS signal
```

6.7 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 DWORD dwStatus;

if (FT_W32_GetCommModemStatus(ftHandle,&dwStatus)) {
       // FT_W32_GetCommModemStatus ok
       if (dwStatus & MS_CTS_ON)
              ; // CTS is on
       if (dwStatus & MS_DSR_ON)
              ; // DSR is on
       if (dwStatus & MS_RI_ON)
              ; // RI is on
       if (dwStatus & MS_RLSD_ON)
              ; // RLSD is on
}
else
       ; // FT_W32_GetCommModemStatus failed
```

6.8 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.

6.9 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; // 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
		; // FT_W32_SetCommState failed
}
else
	; // FT_W32_GetCommState failed
```

6.10 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
FTDCB ftDCB;

if (FT_W32_GetCommState(ftHandle,&ftDCB))
       ; // FT_W32_GetCommState ok, device state is in ftDCB
else
       ; // FT_W32_GetCommState failed
```

6.11 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
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
```

6.12 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.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
FTTIMEOUTS ftTS;

if (FT_W32_GetCommTimeouts(ftHandle,&ftTS))
       ; // FT_W32_GetCommTimeouts OK
else
       ; // FT_W32_GetCommTimeouts failed
```

6.13 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

if (!FT_W32_SetCommBreak(ftHandle))
	; // FT_W32_SetCommBreak failed
else
	; // FT_W32_SetCommBreak OK
```

6.14 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

if (!FT_W32_ClearCommBreak(ftHandle)){
	// FT_W32_ClearCommBreak failed
}
else{
	// FT_W32_ClearCommBreak OK
}
```

6.15 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 to wait for an event to occur.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
DWORD dwMask = EV_CTS | EV_DSR;

if (!FT_W32_SetCommMask(ftHandle,dwMask))
       ; // FT_W32_SetCommMask failed
else
       ; // FT_W32_SetCommMask OK
```

6.16 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)

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 returns events currently being monitored by the device. Event monitoring for these events is enabled by the FT_W32_SetCommMask function.

Example

```
FT_HANDLE ftHandle; // setup by FT_W32_CreateFile
DWORD dwMask;

if (!FT_W32_GetCommMask(ftHandle,&dwMask))
       ; // FT_W32_GetCommMask failed
else
```

```
; // FT_W32_GetCommMask OK
```

6.17 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_SetupComm** (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 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 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 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. 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
DWORD dwEvents;

if (FT_W32_WaitCommEvent(ftHandle, &dwEvents, NULL))
	; // FT_W32_WaitCommEvents OK
else
	; // FT_W32_WaitCommEvents failed
```

2. 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
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
			; // FT_W32_WaitCommEvent OK
			// Events that occurred are stored in dwEvents
	}
}
else {
	// FT_W32_WaitCommEvent OK
	// Events that occurred are stored in dwEvents
}
```

6.18 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

if (FT_W32_PurgeComm(ftHandle,PURGE_TXCLEAR|PURGE_RXCLEAR))
       ; // FT_W32_PurgeComm OK
else
       ; // FT_W32_PurgeComm failed
```

6.19 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 and FT_W32_WriteFile.

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).

6.20 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
COMSTAT newCS;
DWORD dwErrors;
BOOL bChanged = FALSE;

if (!FT_W32_ClearCommError(ftHandle, &dwErrors, (FTCOMSTAT *)&newCS))
       ; // FT_W32_ClearCommError failed

if (dwErrors != dwOldErrors) {
       bChanged = TRUE;
       dwErrorsOld = 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
}
```

7 Contact Information

Head Office – Glasgow, UK

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

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

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road, NeiHu District, Taipei 114, Taiwan
Tel: +886 2 8797 1330
Fax: +886 2 8751 9737

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

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

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

Branch Office – ShangHai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
ChangNing District,
ShangHai, P.R. China

Tel: +86 (21) 62351596
Fax: +86 (21) 62351595

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

Distributor and Sales Representatives

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

Appendix A - Type Definitions

UCHAR	Unsigned char (1 byte)
PUCHAR	Pointer to unsigned char (4 bytes)
PCHAR	Pointer to char (4 bytes)
DWORD	Unsigned long (4 bytes)
LPDWORD	Pointer to unsigned long (4 bytes)
FT_HANDLE	DWORD

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)

- FT_LIST_NUMBER_ONLY = 0x80000000
- FT_LIST_BY_INDEX = 0x40000000
- FT_LIST_ALL = 0x20000000

Flags (see FT_OpenEx)

- 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

Word Length (see FT_SetDataCharacteristics)

- FT_BITS_8 = 8

FT_BITS_7 = 7

Stop Bits (see FT_SetDataCharacteristics)
FT_STOP_BITS_1 = 0
FT_STOP_BITS_2 = 2

Parity (see FT_SetDataCharacteristics)
FT_PARITY_NONE = 0
FT_PARITY_ODD = 1
FT_PARITY_EVEN = 2
FT_PARITY_MARK = 3
FT_PARITY_SPACE = 4

Flow Control (see FT_SetFlowControlFT_SetFlowControl)
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)
FT_PURGE_RX = 1
FT_PURGE_TX = 2

Notification Events (see FT_SetEventNotification)
FT_EVENT_RXCHAR = 1
FT_EVENT_MODEM_STATUS = 2
FT_EVENT_LINE_STATUS = 4

Modem Status (see FT_GetModemStatus)
CTS = 0x10
DSR = 0x20
RI = 0x40
DCD = 0x80

Line Status (see FT_GetModemStatus)
OE = 0x02
PE = 0x04
FE = 0x08
BI = 0x10

Bit Modes (see FT_SetBitMode)
Reset = 0x00
Asynchronous Bit-Bang = 0x01
MPSSE = 0x02
Synchronous Bit-Bang = 0x04
MCU Host Bus Emulation = 0x08
Fast Opto-Isolated Serial Mode = 0x10
CBUS Bit-Bang = 0x20
Single Channel Synchronous 245 FIFO Mode = 0x40

FT232R CBUS EEPROM OPTIONS - Ignored for FT245R (see FT_EE_Program and FT_EE_Read)

```
        CBUS_TXDEN = 0x00
        CBUS_PWRON = 0x01
        CBUS_RXLED = 0x02
        CBUS_TXLED = 0x03
        CBUS_TXRXLED = 0x04
        CBUS_SLEEP = 0x05
        CBUS_CLK48 = 0x06
        CBUS_CLK24 = 0x07
        CBUS_CLK12 = 0x08
        CBUS_CLK6 = 0x09
        CBUS_IOMODE = 0x0A
        CBUS_BITBANG_WR = 0x0B
        CBUS_BITBANG_RD = 0x0C
```

FT_DEVICE_LIST_INFO_NODE (see FT_GetDeviceInfoList and FT_GetDeviceInfoDetail)

```
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)

```
        FT_FLAGS_OPENED = 0x00000001
        FT_FLAGS_HISPEED = 0x00000002
```

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
                           //      1 = FT2232 extensions
                           //      2 = FT232R extensions
                           //      3 = FT2232H extensions
                           //      4 = FT4232H 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;               // not implemented – set to 0
UCHAR IsoInB;               // not implemented – set to 0
UCHAR IsoOutA;              // not implemented – set to 0
UCHAR IsoOutB;              // not implemented – set to 0
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 – this should always be 64
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
        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 designs
        //
        // 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
} FT_PROGRAM_DATA, *PFT_PROGRAM_DATA;
```

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 - Revision History

Revision History

Version 1.00	Initial release in new format.	
	Includes all functions in CDM driver 2.04.06	October, 2008
Version 1.01	Includes FT4232H and FT2232H extensions.	
	Updated addresses.	January, 2009
Version 1.02	Page 65 – removed FT232R and FT245R reference from MCU host emulation and Fast opto modes.	January, 2010