# FT8U232AM / FT8U245AM

## High Speed USB – Legacy Serial and FIFO Interface

## FTD2XX Direct Driver Programmer's Guide

## 27th July 2001


## Future Technology Devices International Ltd.

**St. George's Studios**

**93/97 St. George's Road**

**Glasgow G3 6JA**

**Scotland UK**

**http://www.ftdichip.com**

# Introduction

An FTD2XX device is an FT8U232 or FT8U245 running with FTDI's direct driver FTD2XX.SYS. The device has a programming interface exposed by the dynamic link library FTD2XX.DLL, and this document describes that interface.

# Overview

Before the device can be accessed, it must first be opened. **FT_Open** and **FT_OpenEx** return a handle which is used by all functions in the programming interface to identifiy the device. When the device has been opened successfully, I/O can be performed using **FT_Read** and **FT_Write**. When operations are complete, the device is closed using **FT_Close**.

Functions are available to reset the device (**FT_ResetDevice**); purge receive and transmit buffers (**FT_Purge**); set receive and transmit timeouts (**FT_SetTimeouts**); get receive queue status (**FT_GetQueueStatus**); get device status (**FT_GetStatus**); set and reset break condition (**FT_SetBreakOn**, **FT_SetBreakOff**); set conditions for event notification (**FT_SetEventNotification**).

**FT_ListDevices** returns information about the devices currently connected.

For FT8U232 devices, functions are available to set the baud rate (**FT_SetBaudRate**), and set a non-standard baud rate (**FT_SetDivisor**); set the data characteristics such as word length, stop bits and parity (**FT_SetDataCharacteristics**); set hardware or software handshaking (**FT_SetFlowControl**); set modem control signals (**FT_SetDTR**, **FT_ClrDTR**, **FT_SetRTS**, **FT_ClrRTS**); get modem status (**FT_GetModemStatus**); set special characters such as event and error characters (**FT_SetChars**).

# Reference

The functions which comprise the FTD2XX programming interface are described in this section. Excerpts from the header file FTD2XX.H are included to explain any references in the descriptions of the functions below.

### FT_HANDLE

```
typedef  DWORD  FT_HANDLE
```

### FT_STATUS

```
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
```

### Flags (see FT_OpenEx)

```
FT_OPEN_BY_SERIAL_NUMBER = 1
FT_OPEN_BY_DESCRIPTION = 2
```

### Flags (see FT_ListDevices)

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

### 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_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_ListDevices

**Description**

Get information concerning the devices currently connected. This function can return such information as the number of devices connected, and device strings such as serial number and product description.

**Syntax** FT_STATUS **FT_ListDevices** ( PVOID *pvArg1*,PVOID *pvArg2*, DWORD *dwFlags* )

**Parameters**

*pvArg1* PVOID: Meaning depends on *dwFlags*

*pvArg2* PVOID: Meaning depends on *dwFlags*

*dwFlags* DWORD: Determines format of returned information

**Return Value** FT_STATUS: 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.

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 string information. If *FT_OPEN_BY_SERIAL_NUMBER* bit is set in *dwFlags*, the serial number string will be returned from this function. If *FT_OPEN_BY_DESCRIPTION* bit is set in *dwFlags*, the product description string will be returned from this function. If neither 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* bit is 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* bit is 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.

**Examples**

Sample code shows how to get the number of devices currently connected.

```
FT_STATUS ftStatus;
DWORD numDevs;

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
}
```

This sample shows how to get the serial number of the first device found. 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.

```
FT_STATUS ftStatus;
DWORD devIndex = 0;
char Buffer[16];

ftStatus =
FT_ListDevices((PVOID)devIndex,Buffer,FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_N
UMBER);
if (FT_SUCCESS(ftStatus)) {
    // FT_ListDevices OK, serial number is in Buffer
}
else {
    // FT_ListDevices failed
}
```

This sample shows how to get the product descriptions of all the devices currently connected.

```
FT_STATUS ftStatus;
char *BufPtrs[3];             // pointer to array of 3 pointers
char Buffer1[64];             // buffer for product description of first
device found
char Buffer2[64];             // buffer for product description of second
device
DWORD numDevs;

// 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 (FT_SUCCESS(ftStatus)) {
    // FT_ListDevices OK, product descriptions are in Buffer1 and Buffer2,
and
    // numDevs contains the number of devices connected
}
else {
    // FT_ListDevices failed
}
```

## FT_Open

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

**Syntax** FT_STATUS **FT_Open** ( PVOID *pvDevice*, FT_HANDLE **ftHandle* )

**Parameters**

*pvDevice* ULONG: Must be 0 if only one device is attached. For multiple devices 1, 2 etc.

*ftHandle* FT_HANDLE *: 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_STATUS: 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 *pvDevice* to 0, 1, 2 etc. there is no ability to open a specific device. To open named devices, use the function **FT_OpenEx**.

**Example**

This sample shows how to open a device.

```
FT_HANDLE ftHandle;
FT_STATUS ftStatus;
ftStatus = FT_Open((PVOID) 0,&ftHandle);
if (ftStatus == FT_OK) {
    // FT_Open OK, use ftHandle to access device
}
else {
    // FT_Open failed
}
```

## FT_OpenEx

**Description**

Open the named device and return a handle which will be used for subsequent accesses. The device name can be its serial number or device description.

**Syntax** FT_STATUS **FT_OpenEx** ( PVOID *pvArg1*, DWORD *dwFlags*, FT_HANDLE **ftHandle* )

**Parameters**

*pvArg1* PVOID: Meaning depends on *dwFlags*, but it will normally be interpreted as a pointer to a null terminated string.

*dwFlags* DWORD: FT_OPEN_BY_SERIAL_NUMBER or FT_OPEN_BY_DESCRIPTION.

*ftHandle* FT_HANDLE *: 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_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

**Remarks**

This function should be used to open multiple devices. Multiple devices can be opened at the same time if they can be distinguished by serial number or device description.

**Example**

These samples show how to open two devices simultaneously.

Suppose one device has serial number "FT000001", and the other has serial number "FT999999".

```
FT_STATUS ftStatus;
FT_HANDLE ftHandle1;
FT_HANDLE ftHandle2;

ftStatus = FT_OpenEx("FT000001",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle1);
if (ftStatus == FT_OK) {
        // FT_OpenEx OK, device with serial number "FT000001" is open
}
else {
```

```
        // FT_OpenEx failed
}

ftStatus = FT_OpenEx("FT999999",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle2);
if (ftStatus == FT_OK) {
        // FT_OpenEx OK, device with serial number "FT999999" is open
}
else {
        // FT_OpenEx failed
}
```

Suppose one device has description "USB HS SERIAL CONVERTER", and the other has description "USB PUMP CONTROLLER".

```
FT_STATUS ftStatus;
FT_HANDLE ftHandle1;
FT_HANDLE ftHandle2;

ftStatus = FT_OpenEx("USB HS SERIAL
CONVERTER",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
if (ftStatus == FT_OK) {
        // FT_OpenEx OK, device with description "USB HS SERIAL CONVERTER"
is open
}
else {
        // FT_OpenEx failed
}

ftStatus = FT_OpenEx("USB PUMP
CONTROLLER",FT_OPEN_BY_DESCRIPTION,&ftHandle2);
if (ftStatus == FT_OK) {
        // FT_OpenEx OK, device with description "USB PUMP CONTROLLER" is
open
}
else {
        // FT_OpenEx failed
}
```

## FT_Close

**Description** Close an open device.

**Syntax** FT_STATUS **FT_Close** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to close.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_Read

**Description** Read data from the device.

**Syntax** FT_STATUS **FT_Read** ( FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToRead*, LPDWORD *lpdwBytesReturned* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to read.

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

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

*lpdwBytesReturned* LPDWORD: Pointer to a variable of type DWORD which receives the number of bytes read from the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_Write

**Description** Write data to the device.

**Syntax** FT_STATUS **FT_Write** ( FT_HANDLE *ftHandle*, LPVOID *lpBuffer*, DWORD *dwBytesToWrite*, LPDWORD *lpdwBytesWritten* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to write.

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

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

*lpdwBytesWritten* LPDWORD: Pointer to a variable of type DWORD which receives the number of bytes written to the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_ResetDevice

**Description** This function sends a reset command to the device.

**Syntax** FT_STATUS **FT_ResetDevice** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to reset.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetBaudRate

**Description** This function sets the baud rate for the device.

**Syntax** FT_STATUS **FT_SetBaudRate** ( FT_HANDLE *ftHandle,* DWORD *dwBaudRate* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*dwBaudRate* DWORD: Baud rate.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetDivisor

**Description**

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

**Syntax** FT_STATUS **FT_SetDivisor** ( FT_HANDLE *ftHandle*, USHORT *usDivisor* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*usDivisor* USHORT: Divisor.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

**Remarks**

The application note "Setting Baud rates for the FT8U232AM", which is available on our web site http://www.ftdichip.com , describes how to calculate the divisor for a non standard baud rate.

**Example**

Suppose we want to set a baud rate of 5787 baud. A simple calculation suggests that a divisor of 4206 should work.

```
FT_HANDLE ftHandle;    // handle of device obtained from FT_Open or
FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_SetDivisor(ftHandle,0x4206);
if (ftStatus == FT_OK) {
        // FT_SetDivisor OK, baud rate has been set to 5787 baud
}
else {
        // FT_SetDivisor failed
}
```

## FT_SetDataCharacteristics

**Description** This function sets the data characteristics for the device.

**Syntax** FT_STATUS **FT_SetDataCharacteristics** ( FT_HANDLE *ftHandle,* UCHAR *uWordLength,* UCHAR *uStopBits,*UCHAR *uParity* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*uWordLength* UCHAR: Number of bits per word - must be FT_BITS_8 or FT_BITS_7.

*uStopBits* UCHAR: Number of stop bits - must be FT_STOP_BITS_1 or FT_STOP_BITS_2.

*uParity* UCHAR: FT_PATITY_NONE, FT_PARITY_ODD, FT_PARITY_EVEN, FT_PARITY_MARK, FT_PARITY_SPACE.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetFlowControl

**Description** This function sets the flow control for the device.

**Syntax** FT_STATUS **FT_SetFlowControl** ( FT_HANDLE *ftHandle*, USHORT us*FlowControl*, UCHAR u*Xon*,UCHAR u*Xoff* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*usFlowControl* USHORT: Must be one of FT_FLOW_NONE, FT_FLOW_RTS_CTS, FT_FLOW_DTR_DSR, FT_FLOW_XON_XOFF

*uXon* UCHAR: Character used to signal XON. Only used if flow control is FT_FLOW_XON_XOFF.

*uXoff* UCHAR: Character used to signal XOFF. Only used if flow control is FT_FLOW_XON_XOFF.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetDTR

**Description** This function sets DTR.

**Syntax** FT_STATUS **FT_SetDTR** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_ClrDTR

**Description** This function clears DTR.

**Syntax** FT_STATUS **FT_ClrDTR** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetRTS

**Description** This function sets RTS.

**Syntax** FT_STATUS **FT_SetRTS** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_ClrRTS

**Description** This function clears RTS.

**Syntax** FT_STATUS **FT_ClrRTS** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_GetModemStatus

**Description** Gets the modem status from the device.

**Syntax** FT_STATUS **FT_GetModemStatus** ( FT_HANDLE *ftHandle*, LPDWORD *lpdwModemStatus* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to read.

*lpdwModemStatus* LPDWORD: Pointer to a variable of type DWORD which receives the modem status from the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetChars

**Description** This function sets the special characters for the device.

**Syntax** FT_STATUS **FT_SetChars** ( FT_HANDLE *ftHandle*, UCHAR *uEventCh*, UCHAR *uEventChEn*, UCHAR *uErrorCh*, UCHAR *uErrorChEn* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*uEventCh* UCHAR: Event character,

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

*uErrorCh* UCHAR: Error character,

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

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_Purge

**Description** This function purges the receive and transmit buffers in the device.

**Syntax** FT_STATUS **FT_Purge** ( FT_HANDLE *ftHandle*, DWORD *dwMask* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*dwMask* DWORD: Any combination of FT_PURGE_RX and FT_PURGE_TX.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetTimeouts

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

**Syntax** FT_STATUS **FT_SetTimeouts** ( FT_HANDLE *ftHandle*, DWORD *dwReadTimeout*, DWORD *dwWriteTimeout* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*dwReadTimeout* DWORD: Read timeout in milliseconds.

*dwWriteTimeout* DWORD: Write timeout in milliseconds.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_GetQueueStatus

**Description** Gets the number of characters in the receive queue.

**Syntax** FT_STATUS **FT_GetQueueStatus** ( FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to read.

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

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetBreakOn

**Description** Sets the BREAK condition for the device.

**Syntax** FT_STATUS **FT_SetBreakOn** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_SetBreakOff

**Description** Resets the BREAK condition for the device.

**Syntax** FT_STATUS **FT_SetBreakOff** ( FT_HANDLE *ftHandle* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

**Return Value** FT_STATUS: FT_OK if successful, otherwise the return value is an FT error code.

## FT_GetStatus

**Description**

Gets the device status including number of characters in the receive queue, number of characters in the transmit queue, and the current event status.

**Syntax** FT_STATUS **FT_GetStatus** ( FT_HANDLE *ftHandle*, LPDWORD *lpdwAmountInRxQueue*, LPDWORD *lpdwAmountInTxQueue*, LPDWORD *lpdwEventStatus* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device to read.

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

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

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

**Return Value** FT_STATUS: 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*.

## FT_SetEventNotification

**Description** Sets conditions for event notification.

**Syntax** FT_STATUS **FT_SetEventNotification** ( FT_HANDLE *ftHandle*, DWORD *dwEventMask*, PVOID *pvArg* )

**Parameters**

*ftHandle* FT_HANDLE: handle of the device.

*dwEventMask* DWORD: conditions that cause the event to be set.

*pvArg* PVOID: interpreted as a handle of an event

**Return Value** FT_STATUS: 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.

**Example**

This example 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 which 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
}
```