Future Technology Devices International Ltd.

Application Note AN_133

D2XX_Access_Using PERL_Interface

PERL 5.0 Programmers Guide

Document Reference No.: FT_0000199

Version 1.0

Issue Date: 2009-11-10

This document provides the application programming interface (API) for the FTD2XX DLL using the Perl programming language.

Future Technology Devices International Limited (FTDI)

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

Table of Contents

1 Preface 3
2 Overview 4
2.1 Prerequisites 4
2.2 Installing Win32::FTDI::FTD2XX 4
3 Perl D2XX Functions 6
3.1 New 6
3.2 GetNumDevices 6
3.3 OpenByIndex 6
3.4 OpenBySerial 7
3.5 GetDeviceInfo 7
3.6 Close 8
3.7 CyclePort 8
3.8 SetBaudRate 8
3.9 SetDataCharacteristics 9
3.10 SetFlowControl 9
3.11 SetDtr 10
3.12 ClrDtr 10
3.13 SetRts 11
3.14 ClrRts 11
3.15 SetTimeouts 11
3.16 GetTimeouts 12
3.17 SetReadTimeout 12
3.18 SetWriteTimeout 13
3.19 GetReadTimeout 13
3.20 GetWriteTimeout 13
3.21 GetModemStatus 14
3.22 WaitForModem 14
3.23 GetQueueStatus 15
3.24 Read 15
3.25 Write 15
3.26 ResetDevice 16
3.27 ResetPort 16
3.28 SetBreakOn 17

3.29 SetBreakOff 17
3.30 GetStatus 17
3.31 Purge 18
3.32 SetChars 18
3.33 Rescan 19
3.34 Reload 19
3.35 StopInTask 20
3.36 RestartInTask 20
3.37 SetLatencyTimer 20
3.38 GetLatencyTimer 21
3.39 GetBitMode 21
3.40 SetBitMode 22
3.41 SetUSBParameters 22
4 Code Example 23
5 Contact Information 25
Appendix A - References 27
Appendix B - Revision History 28
Revision Record Sheet 29

1 Preface

The D2XX interface is a proprietary interface specifically for FTDI devices. This document will provide an explanation of how this interface can be accessed using the Perl5 programming language via the Win32::FTDI::FTD2XX wrapper.

The Perl D2XX wrapper was kindly provided by Scott K. MacPherson and is free software licensed under the terms of Perl itself allowing for redistribution and/or modification.

The code examples contained within this document are for demonstration purposes only and FTDI extend no responsibility or guarantees regarding the correctness of this code.

2 Overview

FTDI's D2XX library provides an interface to FTDI's USB-UART and USB-FIFO ICs, this library provides additional functions that are not available with standard Windows COM port APIs. Due to the nature of the Perl programming language it is not possible to directly communicate using the D2XX library. The Win32::FTDI::FTD2XX wrapper provides an abstraction between the FTDI D2XX library and Perl. This document will provide an overview for installing the Win32::FTDI::FTD2XX wrapper and also act as a reference for the available functions for interfacing with an FTDI device via the D2XX library.

2.1 Prerequisites

To use the Perl D2XX wrapper you must first have ActivePerl v5.8 or greater installed on your system. The latest version (v5.10 at time of writing) of this can be found on the ActiveState website. The FTDI FTD2XX driver must also be installed, the latest version of the D2XX Driver is available from the FTDI website currently at version 2.04.16. Finally, an FTDI device to communicate with is required.

2.2 Installing Win32::FTDI::FTD2XX

Perl provides extensions of the language by allowing users to install Perl modules. The Win32::FTDI::FTD2XX is provided to the user as an additional module that must be installed on the system prior to calling any of the functions contained within it.

Installing Perl modules can be a difficult and time consuming process; below is one option that may adopted to install the module but there are possibly other avenues to explore not outlined within. There are plenty of websites available on the internet that give detailed descriptions of installing Perl modules.

The Win32::FTDI::FTD2XX package is available as a free download from the CPAN website. Download the latest version, 1.04, and save the file within the Perl directory of your machine. The subsequent file should have the extension tar.gz, this is a compression format that can be unzipped using third party extraction tools, e.g. WinZip but there are other alternatives available free from the internet. Extract the files into the Perl directory on your machine, in our case it was '*C:\Perl*'.

The next step is to install nmake on the system; this is available as a free download from the Microsoft Website. Save the file when prompted and run the executable Nmake15.exe. This will extract 3 files into the current directory: NMAKE.ERR; NMAKE.EXE and README.TXT. Paste the files NMAKE.ERR and NMAKE.EXE into the bin folder within your Perl directory 'C:\Perl\Bin'.

Back in the Perl directory containing the unzipped Perl FTD2XX package, open the file MAINFEST with notepad to view its contents. This manifest gives a list of the all the files that should have been included with the tar.gz download.

Move the files that were downloaded so that the paths of these files match those within the manifest file. For example, cut the file Win32-FTDI-FTD2XX.t and paste it into the folder 'C:\Perl\t', cut instp5.dll and paste in the directory 'C:\Perl\bin'. Complete this for the remaining 2 files bearing in mind that there needs to be a new folder called 'FTDI' created within the Win32 directory.

```
t/Win32-FTDI-FTD2XX.t
bin/instp5dll.pl
lib/Win32/FTDI/FTD2XX.pm
lib/Win32/FTDI/p5ftd2xx.dll
```

The module may now be installed; click start->run, type '*cmd*' and press OK. This will launch the command prompt window.

Browse to the Perl folder within the windows directory using the command prompt window, in this case:

```
CD C:\Perl
```

Now type '*perl Makefile.pl',* this command will check to make sure that all the module files are present within their correct respective directories. If an error is encountered stating that some or all of the files are missing then make sure that the module files have been copied into the correct directories as stated in the MANIFEST file above.

Still within the command window type the following commands to install the module:

```
nmake
nmake test
nmake install
```

The Perl module should now be installed, to test that the module is available and working, create a new text file called ftd2xx.pl and paste the following code into it.

```
#!/usr/bin/perl

use Win32::FTDI::FTD2XX
```

Browse to the file, again with the command prompt window, and run the application by typing `ftd2xx.pl`. The code should run with no compilation errors indicating that the module has been installed correctly and is now available for use.

3 Perl D2XX Functions

3.1 New

Summary

This command creates a new instance of the WIN32::FTDI class; this class instance is then used throughout the program to reference the device. This function must be called at the start of the application to allow communication to a device. The new function has a built in auto-destroy feature that will close a handle to any device as part of Perl's garbage collection when the program ends.

Definition

```
Win32::API::FTD2XX->new()
```

Parameters

None

Return Value

Returns an object instance that is used throughout the program to reference to an FTDI devices.

Example

```
my $FTDIdevice = Win32::API::FTD2XX->new();
....
my $numDevices = $FTDIdevice->GetNumDevices();
....
```

3.2 GetNumDevices

Summary

Returns the number of FTDI devices that are currently connected to the host machine.

Definition

GetNumDevices()

Parameters

None

Return Value

Upon successful completion of the function it will return the number of devices connected to the host, otherwise NULL.

Example

```
my $FTDIdevice = Win32::API::FTD2XX->new();
my $numDevices = $FTDIdevice->GetNumDevices();# Get the number of connected devices.
```

3.3 OpenByIndex

Summary

Open an FTDI device based on the index ID with which it has been enumerated on the host machine.

Definition

OpenByIndex($devIndex)

Parameters

$devIndex – an integer indicating the index of the device.

Return Value

Returns true if a handle to the specified device has been obtained, false otherwise.

Example

```
my $FTDIdevice = Win32::API::FTD2XX->new();
my $devToOpen = 0;
my $devOpen = FTDIdevice->OpenByIndex($devToOpen);
if($devOpen)
{
   print "Device $devToOpen is open!";
}
```

3.4 OpenBySerial

Summary

Open an FTDI device based on the serial ID of the chip.

Definition

OpenBySerial($devSerial)

Parameters

$devSerial – Device serial number.

Return Value

Returns true if a handle to the specified device has been obtained, false otherwise.

Example

```
my $FTDIdevice = Win32::API::FTD2XX->new();
my $devToOpen = "FT12345";
my $devOpen = FTDIdevice->OpenBySerial($devToOpen);
if($devOpen)
{
   print "Device $devToOpen is open!";
}
```

3.5 GetDeviceInfo

Summary

Used to obtain string descriptors for an **open** device.

Definition

GetDeviceInfo()

Parameters

None

Return Value

String descriptors for the device are returned in the form of a hash, if the function fails to complete it will return NULL. Please refer to the example below for further details of the hash.

Example

```
my $FTDIdevice = Win32::API::FTD2XX->new();
my $devInfo = $ FTDIdevice ->GetDeviceInfo();
if( $devInfo )
{
   my $out = sprintf( "  Type:\t\t%d (%s)\n  ID:\t\tVID(%04X) PID(%04X)\n
   Serial:\t%s\n  Descr:\t%s\n",$devInfo->{TypeID}, $devInfo->{TypeNm}, $devInfo-
   >{VID}, $devInfo->{PID}, $devInfo->{Serial}, $devInfo->{Descr} );
   print "$out";
```

```
}
```

3.6 Close

Summary

Closes the handle to any open FTDI device. Please note that there is an auto-destory feature that will close the handle to an open FTDI device as part of the Perl garbage collection.

Definition

Close()

Parameters

None

Return Value

This function will return true if a device has been closed successfully, false otherwise.

Example

```
my $devClosed = $FTDIdevice->Close();
if($devCLosed)
{
        # Operation successful
}
```

3.7 CyclePort

Summary

Send a cycle command to the USB port, the effect of this function is the same as disconnecting and reconnecting the device. This can be useful when recovering from fatal errors or forcing the FTDI to read the contents of the EEPROM again. Please note that there is an unspecified wait time for performing this function until the FTDI is stable again. Attempting to communicate during this time may result in an invalid handle from the device.

Definition

CyclePort()

Parameters

None

Return Value

This function will return true on successful completion of the operation, false otherwise.

Example

```
my $result = $FTDIdevice->CyclePort();
if($result)
{
        # Operation Successful
}
```

3.8 SetBaudRate

Summary

Sets the baud rate of the currently open device.

Definition

SetBaudRate($baudRate)

Parameters

$baudRate

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $baudRate = 115200;
my $status = $FTDIdevice->SetBaudRate($baudRate);
if($status)
{
	# Operation Successful
}
```

3.9 SetDataCharacteristics

Summary

Sets the data characteristics of the currently open device. Please refer to the D2XX Programmer's Guide for a more detailed description of the parameters.

Definition

SetDataCharacteristics($dataBits, $stopBits, $parityBits)

Parameters

$dataBits – the number of bits per word, must be 8 or 7

$stopBits – the number of stop bits, must be 1 or 2

$parityBits - parity

Return Value

Returns true on successful completion, false otherwise.

Example

```
use constant {
DATABITS => 8,
STOPBITS => 1,
PARITYBITS => 0
};
....
my $status = $FTDIdevice->SetDataCharacteristics(DATABITS, STOPBITS, PARITYBITS);
if($status)
{
	# Operation Successful
}
```

3.10 SetFlowControl

Summary

Sets the flow control for the currently open device. Please refer to the D2XX Programmer's Guide for a more detailed description of the parameters.

Definition

SetFlowControl($flowControl, $Xon, $Xoff)

Parameters

$flowControl

$Xon

$Xoff

Return Value

Returns true on successful completion, false otherwise.

Example

```
use constant {
FLOWCTRL => 256,
XON => 0,
XOFF => 0
};
....
my $status = $FTDIdevice->SetFlowControl(FLOWCTRL, XON, XOFF);
if($status)
{
	# Operation Successful
}
```

3.11 SetDtr

Summary

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

Definition

SetDtr()

Parameters

None

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $status = $FTDIdevice->SetDtr();
if($status)
{
	# Operation Successful
}
```

3.12 ClrDtr

Summary

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

Definition

ClrDtr()

Parameters

None

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $status = $FTDIdevice->ClrDtr();
if($status)
{
        # Operation Successful
}
```

3.13 SetRts

Summary

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

Definition

SetRts()

Parameters

None

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $status = $FTDIdevice->SetRts();
if($status)
{
        # Operation Successful
}
```

3.14 ClrRts

Summary

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

Definition

ClrRts()

Parameters

None

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $status = $FTDIdevice->ClrRts();
if($status)
{
        # Operation Successful
}
```

3.15 SetTimeouts

Summary

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

Definition

SetTimouts($readTimeout, $writeTimeout)

Parameters

$readTimeout

$writeTimeout

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $status = $FTDIdevice->SetTimeouts(5000, 1000);# Read 5 secs, write 1 sec
if($status)
{
        # Operation Successful
}
```

3.16 GetTimeouts

Summary

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

Definition

GetTimeouts()

Parameters

None

Return Value

Returns the previously set read and write timeouts for the device, will return null if the operation has been unsuccessful.

Example

```
(my $readTimeout, my $writeTimeout) = $FTDIdevice->GetTimeouts();
print("Read Timeout: $readTimeout\n");
print("Write Timeout: $writeTimeout\n");
```

3.17 SetReadTimeout

Summary

This function sets the read timeout for the device.

Definition

SetReadTimeout($readTimeout)

Parameters

$readTimeout

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $readTimeout = 1000;
my $success = $FTDIdevice->SetReadTimeout($readTimeout);
if($success)
{
        # Operation Successful
}
```

3.18 SetWriteTimeout

Summary

This function sets the write timeout for the device.

Definition

SetWriteTimeout($writeTimeout)

Parameters

$writeTimeout

Return Value

Returns true on successful completion, false otherwise.

Example

```
my $writeTimeout = 1000;
my $success = $FTDIdevice->SetWriteTimeout($writeTimeout);
if($success)
{
        # Operation Successful
}
```

3.19 GetReadTimeout

Summary

This function gets the read timeout for the device.

Definition

GetReadTimeout()

Parameters

None

Return Value

If the operation is successful it will return the read timeout for the current device, null otherwise.

Example

```
my $readTimeout = $FTDIdevice->GetReadTimeout();
if($readTimeout)
{
        print("Read timeout: $readTimeout");
}
```

3.20 GetWriteTimeout

Summary

This function gets the write timeout for the device.

Definition

GetWriteTimeout()

Parameters

None

Return Value

If the operation is successful it will return the write timeout for the current device, null otherwise.

Example

```
my $writeTimeout = $FTDIdevice->GetWriteTimeout();
if($writeTimeout)
{
	print("Write timeout: $writeTimeout");
```

3.21 }GetModemStatus

Summary

Gets the modem status and line status from the device. Please refer to the D2XX Programmer's Guide for a more detailed description of the return value.

Definition

GetModemStatus()

Parameters

None

Return Value

If the operation is successful it will return the modem status for the current device, null otherwise.

Example

```
my $modemStatus = 0;
my $lineStatus = 0;
my $status = $FTDIdevice->GetModemStatus();
if($status)
{
	$lineStatus = (($status >> 8) & 0x000000FF);
	$modemStatus = ($status & 0x000000FF);
}
```

3.22 WaitForModem

Summary

This function can be used to suspend program execution until one or more of the modem status bits is set.

Definition

waitForModem($modemStatusBitmask, $timeout, $pollTm)

Parameters

$modemStatusBitmask – the modem bit to wait on, defined under Modem Status within the D2XX Programmer's Guide appendix A.

$timeout – Optional, the wait time in seconds, if blank the device will wait indefinitely.

$pollTm – Optional, this is the time in seconds between polls of the device, default value is 0.25 secs.

Return Value

Returns true on successful completion, false otherwise.

Example

```
# Waits 3 seconds for the CTS line to be asserted.
my $status = $FTDIdevice->waitForModem(0x10, 3);
if($status)
{
	# Operation successful
}
```

3.23 GetQueueStatus

Summary

Gets the number of bytes that are in the receive queue to be read from the device.

Definition

GetQueueStatus()

Parameters

None

Return Value

If the operation is successful it will return the number of bytes in the read queue for the current device, if there are no bytes to read zero will be returned.

Example

```
my $rxBytesAvail = $FTDIdevice->GetQueueStatus();
if($rxBytesAvail)
{
	print("$rxBytesAvail bytes to be read from the device.");
}
```

3.24 Read

Summary

Read any data from the device.

Definition

Read($rxBytesAvail)

Parameters

$ rxBytesAvail

Return Value

If the operation is successful it will return the number of bytes read ($bytesReturned_p) and the data ($readBuffer_p) from the device as scalar values.

Example

```
my $bytesReturned_p; # packed version of bytes read
my $readBuffer_p; # packed version of read buffer
my $readBuffer; # unpacked version of read buffer

my $rxBytesAvail = $FTDIdevice->GetQueueStatus();
if($rxBytesAvail)
{
	($bytesReturned_p, $readBuffer_p) = $FTDIdevice->Read($rxBytesAvail);
	$readBuffer = unpack("a*", $readBuffer_p);# Unpack the data from the device
	print("Read: $readBuffer from the device.");
}
```

Note

Data returned from the device is in a raw binary format, the unpack function is required to format the data as an ASCII string.

3.25 Write

Summary

Write data to the device.

Definition

write($writeBuffer, $bytesToWrite)

Parameters

$writeBuffer

$bytesToWrite – optional parameter, if this is not specified 'length($writeBuffer)' is used.

Return Value

If the operation is successful it will return the number of bytes that have been written to the device.

Example

```
my $writeBuffer = "Hello World!"; # unpacked version of write buffer
my $bytesToWrite = 12; # Size of the write buffer.
my $writeBuffer_p; # packed version of write buffer
$writeBuffer_p = pack("a*", writeBuffer);# pack the write buffer.

my $bytesWritten = $FTDIdevice->Write($writeBuffer_p, $bytesToWrite);
if($bytesWritten == $bytesToWrite)
{
        print("Data successfully sent to the device.");
}
```

3.26 ResetDevice

Summary

Sends a reset command to the device.

Definition

ResetDevice()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->ResetDevice();
if($success)
{
        # Device Reset
}
```

3.27 ResetPort

Summary

Sends a reset command to the port.

Definition

ResetPort()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->ResetPort();
if($success)
{
	# Port Reset successful.
}
```

3.28 SetBreakOn

Summary

Sets the break condition for the device.

Definition

SetBreakOn()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->SetBreakOn();
if($success)
{
	# Operation successful.
}
```

3.29 SetBreakOff

Summary

Resets the break condition for the device.

Definition

SetBreakOff()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->SetBreakOff();
if($success)
{
	# Operation successful.
}
```

3.30 GetStatus

Summary

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

Definition

GetStatus()

Parameters

None

Return Value

Returns amount of data in the Rx queue, the amount of data in the Tx queue and the current event status of the device. Please refer to the D2XX Programmer's Guide for a more detailed description of the event status messages returned from this function.

Example

```
(my $rxQueue, my $txQueue, my $eventStatus) = $FTDIdevice->GetStatus();
print("Size of receive queue: $rxQueue");
print("Size of transmit queue: $txQueue");
```

3.31 Purge

Summary

Purges receive and transmit buffers in the device.

Definition

Purge($mask)

Parameters

$mask

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
use constant{
FT_PURGE_RX = 1,
FT_PURGE_TX = 2
};
....
# Purge both RX and TX buffers...
my $success = $FTDIdevice->Purge(FT_PURGE_RX | FT_PURGE_TX);
if($success)
{
        # Operation successful.
}
```

3.32 SetChars

Summary

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

Definition

SetChars($eventCh, $eventChEn, $errorCh, $errorChEn)

Parameters

$eventCh – the event character.

$eventChEn – 0 if event character disabled, non-zero otherwise.

$errorCh – the error character.

$errorChEn – 0 if error character disabled, non-zero otherwise.

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->SetChars(0x12, 1 , 0x14, 1);
if($success)
{
	# Operation successful.
}
```

3.33 Rescan

Summary

This function can be used to try to recover a device programmatically. This is the equivalent to clicking "Scan for hardware changes" within the device manager. All connected USB devices are scanned as well as FTDI devices.

Definition

Rescan()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->Rescan();
if($success)
{
	# Operation successful.
}
```

3.34 Reload

Summary

This function forces a reload of the driver for the devices with a specific Vendor ID and Product ID as specified. If the VID and PID parameters are null, all USB devices connected will reload their drivers. Please note that this function will not work correctly on 64-bit Windows when called from a 32-bit application.

Definition

Reload($VID, $PID)

Parameters

$VID – vendor ID, the FTDI default is 0x0403.

$PID – product ID.

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success = $FTDIdevice->Reload(0x0403, 0x6001); # Standard FT232R device.
```

```
if($success)
{
        # Operation successful.
}
```

3.35 StopInTask

Summary

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 in conjunction with the RestartInTask function which gets the IN task running again.

Definition

StopInTask()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $success;
do {
        $success = $FTDIdevice->StopInTask();
} while(!$success)
#
# Do something – for example purge device
#
do {
        $success = $FTDIdevice->RestartInTask();
} while(!$success)
```

3.36 RestartInTask

Summary

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

Definition

RestartInTask()

Parameters

None

Return Value

Returns true if the operation has been successful, false otherwise.

Example

See StopInTask example.

3.37 SetLatencyTimer

Summary

Set the latency timer value. The timeout is programmable and can be set at 1 ms intervals between 2 ms and 255 ms. This allows the device to be better optimized for protocols requiring faster response

times from the short data packets. This function is not available for the FT8U232AM and FT8U245AM devices which have a hardcoded value of 16 ms.

Definition

SetLatencyTimer($timer)

Parameters

$timer – valid range 2 - 255

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $timer = 100;
my $success = $FTDIdevice->SetLatencyTimer($timer);
if($success)
{
        # Operation Successful.
}
```

3.38 GetLatencyTimer

Summary

Get the current value of the latency timer.

Definition

GetLatencyTimer()

Parameters

None

Return Value

Returns the latency timer value if successful, null otherwise.

Example

```
my $timer = $FTDIdevice->GetLatencyTimer();
if($timer)
{
        # Operation Successful.
}
```

3.39 GetBitMode

Summary

Get the instantaneous value of the data bus. Please refer to the D2XX Programmer's Guide for a more detailed description of this function.

Definition

GetBitMode()

Parameters

None

Return Value

Returns the instantaneous value of the data bus if successful, null otherwise.

Example

```
my $mode = $FTDIdevice->GetBitMode();
```

3.40 SetBitMode

Summary

Get the current value of the latency timer. Please refer to the D2XX Programmer's Guide for a more detailed description of this function.

Definition

SetBitMode($mode)

Parameters

$mode – see the D2XX Programmer's Guide

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $mode = 1; # Asynchronous bit-bang mode.
my $success = $FTDIdevice->SetBitMode($mode);
if($success)
{
	# Operation successful.
}
```

3.41 SetUSBParameters

Summary

Set the USB request transfer size. This function can be used to change the transfer sizes from the default 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 64 kbytes.

Definition

SetUSBParameters($inTransferSize, $outTransferSize);

Parameters

$inTransferSize

$outTransferSize

Return Value

Returns true if the operation has been successful, false otherwise.

Example

```
my $inTransferSize = 16384;
my $status = $FTDIdevice->SetUSBParameters(inTransferSize, 0);
if($status)
{
	# Operation Successful
}
```

4 Code Example

LoopBack

Below is a short example of a loopback test using the Perl D2XX wrapper. The program will list all the devices connected to the host and allow the user to specify which device to open. If a connection to a device has been established the program will attempt to send 'Hello World!', then listen for a loopback from the device. To run this application you will need an FTDI device will a loopback connection fitted. Paste the code into a text editor and save this as `loopback.pl`.

```
#=============================================
#!/usr/bin/perl
# Demonstrate loopback test
#=============================================

use strict;
use Win32::FTDI::FTD2XX;
use POSIX;

my $FTDIdevice = Win32::FTDI::FTD2XX->new(); # Create a new instance of the P5D2XX class

# List all the connected devices
print "Checking for connected devices...\n";
my $numDevices = $FTDIdevice->GetNumDevices();
  if($numDevices)
  {
    print "Found $numDevices device(s) connected to the host.\n";
    for(my $i = 0 ; $i < $numDevices ; $i++)
    { # Loop through each of the connected devices.
      my $devOpen = $FTDIdevice->OpenByIndex( $i );
      # Get device information for that device.
      my $devInfo = $FTDIdevice->GetDeviceInfo(1);
      if( $devInfo )
      {
        print "\n---------Device $i--------\n";
        my $out = sprintf( "  Type:\t\t%d (%s)\n  ID:\t\tVID(%04X) PID(%04X)\n
        Serial:\t%s\n  Descr:\t%s\n",
        $devInfo->{TypeID}, $devInfo->{TypeNm}, $devInfo->{VID}, $devInfo->{PID},
        $devInfo->{Serial}, $devInfo->{Descr} );
        print "$out";
        print "=========================\n";
      }
      else
      {
        print "Didnt print data\n\n\n";
        return 0;
      }
      my $devClose = $FTDIdevice->Close();
```

```
    }
    # Connect to one of the devices.
    print "\n";
    print "Connect to device number? ";
    my $devNum = getchar(); # gets the value from the command line
    print "\n";
    print "Opening device...\n";
    my $devOpen = $FTDIdevice->OpenByIndex( $devNum );
    if( $devOpen == 1)
    {
      print "Connected!\n";
    }
 }

# Send some data to the device
print "Sending 'Hello World!' to the device\n";
my $writeBuffer = "Hello World!";
my $writeBuffer_p = pack("a*", $writeBuffer); # Creates a packed version of the buffer.
my $bytesWritten = $FTDIdevice->Write($writeBuffer_p);
# Check to see if all the data has been sent
if($bytesWritten == length($writeBuffer))
{
    print "Sent all the data\n";
}
sleep(1);
# Check to see if there is any data to read
my $bytesAvail = $FTDIdevice->GetQueueStatus();
# Read any data from the chip
if($bytesAvail)
{
  (my $bytesRead_p, my $readBuffer_p) = $FTDIdevice->Read($bytesAvail);
   my $readBuffer = unpack("a$bytesRead_p", $readBuffer_p);
   print "Read $bytesRead_p bytes from the device\n";
   print "Read $readBuffer back from the device\n";
   if($writeBuffer == $readBuffer)
   {
      print "Successful!\n";
   }
   else
   {
      print "Failed, data did not match!";
   }
}
```

5 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
Web Shop URL http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

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

Branch Office – Hillsboro, Oregon, USA

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.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,
Shanghai, 200051
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 - References

Active State website: http://www.activestate.com/activeperl/

CPAN website: http://search.cpan.org/~skmacphe/

Download for nmake:

http://download.microsoft.com/download/vc15/Patch/1.52/W95/EN-US/Nmake15.exe

Appendix B - Revision History

Revision History

Version draft	Initial draft	Oct, 2009
Version 1.0	Revision 1	10/11/09