Technical Document

PyD3XX Programmer's Guide

Version 1.0.8

Release Date: August 21, 2025

PyD3XX is a Python wrapper created by Hector Soto for FTDI's D3XX library. This technical document gives an overview of PyD3XX's Application Programming Interface (API).

Author: Hector Soto
Email: hector.soto@ftdichip.com
PyPi Repo: PyD3XX · PyPI
GitHub Repo: https://github.com/S-Hector/PyD3XX

Table of Contents

1 – Introduction	**4**
1.1 – Driver Library Files	4
1.2 – Interfaces, Channels, & Endpoints	5
1.3 – SetPrintLevel()	5
2 – PyD3XX Functions	**6**
2.1 – FT_CreateDeviceInfoList()	6
2.2 – FT_GetDeviceInfoList()	7
2.2B – FT_GetDeviceInfoDict()	8
2.3 – FT_GetDeviceInfoDetail()	8
2.4 – FT_ListDevices()	9
2.5 – FT_Create()	11
2.6 – FT_Close()	12
2.7 – FT_WritePipe()	13
2.8 – FT_ReadPipe()	14
2.9 – FT_WritePipeEx()	16
2.9B – FT_WritePipeAsync()	18
2.10 – FT_ReadPipeEx()	21
2.10B – FT_ReadPipeAsync()	22
2.11 – FT_GetOverlappedResult()	25
2.12 – FT_InitializeOverlapped()	27
2.13 – FT_ReleaseOverlapped()	27
2.14 – FT_SetStreamPipe()	28
2.15 – FT_ClearStreamPipe()	29
2.16 – FT_SetPipeTimeout()	30
2.17 – FT_GetPipeTimeout()	31
2.18 – FT_AbortPipe()	32
2.19 – FT_GetDeviceDescriptor()	33
2.20 – FT_GetConfigurationDescriptor()	34
2.21 – FT_GetInterfaceDescriptor()	35
2.21B – FT_GetStringDescriptor()	35
2.22 – FT_GetPipeInformation()	36
2.23 – FT_GetDescriptor()	37
2.24 – FT_ControlTransfer()	38
2.25 – FT_GetVIDPID()	39

2.26 – FT_EnableGPIO() 39
2.27 – FT_WriteGPIO() 40
2.28 – FT_ReadGPIO() 41
2.29 – FT_SetGPIOPull() 41
2.30 – FT_SetNotificationCallback() 42
2.31 – FT_ClearNotificationCallback() 44
2.32 – FT_GetChipConfiguration() 44
2.33 – FT_SetChipConfiguration() 45
2.34 – FT_IsDevicePath() 46
2.35 – FT_GetDriverVersion() 46
2.35B – GetDriverVersion() 47
2.36 – FT_GetLibraryVersion() 48
2.36B – GetLibraryVersion() 48
2.37 – FT_CycleDevicePort() 49
2.37B – FT_ResetDevicePort() 49
2.38 – FT_SetSuspendTimeout() 50
2.39 – FT_GetSuspendTimeout() 51
3 – PyD3XX Classes 52
3.1 – FT_Buffer 52
3.2 – FT_Device 52
3.3 – FT_DeviceDescriptor 53
3.4 – FT_ConfigurationDescriptor 54
3.5 – FT_InterfaceDescriptor 54
3.6 – FT_StringDescriptor 55
3.7 – FT_EndpointDescriptor 55
3.8 – FT_Pipe 56
3.9 – FT_Overlapped 56
3.10 – FT_SetupPacket 57
3.11 – FT_60XCONFIGURATION 57
Appendix A - Miscellaneous 59
Appendix B – Version History 62

1 – Introduction

The PyD3XX Python package is a Python wrapper for Future Technology Devices International's (FTDI's) C language D3XX library. PyD3XX was created so you could easily import a Python API that is cross-compatible with Windows, Linux, & macOS, to interact with FTDI's FT600 and FT601 devices. PyD3XX's API was designed to closely follow the naming conventions of the original D3XX C library's API while requiring zero ctypes knowledge to effectively use PyD3XX.

This programmer's guide provides an overview of PyD3XX's functions and classes.

1.1 – Driver Library Files

PyD3XX comes with multiple D3XX dynamic library files for different operating systems and architectures. PyD3XX will automatically load whichever D3XX dynamic library file is necessary to run on the operating system and architecture it detects. Table 1.1.0 shows what D3XX dynamic library files are loaded for specific operating systems and architectures. For Windows, the older non-WinUSB drivers are designated as "Legacy" in Table 1.1.0.

Operating System			Library Version	Library File
Windows	WinUSB	64-Bit (x64)	1.4.0.1	FTD3XXWU.dll
		32-bit (x86)	1.4.0.1	FTD3XXWU_32.dll
		ARM	1.4.0.1	FTD3XXWU_ARM.dll
	Legacy	64-Bit (x64)	1.3.0.10	FTD3XX.dll
		32-bit (x86)	1.3.0.4	FTD3XX_32.dll
Linux	64-Bit (x64)		1.0.16	libftd3xx.so
	32-bit (x86)		1.0.16	libftd3xx_32.so
	ARM 64-bit		1.0.16	libftd3xx_ARM.so
	ARM 32-bit		1.0.16	libftd3xx_ARM_32.so
macOS	64-Bit (x64)		1.0.16	libftd3xx.dylib
	32-bit (x86)		1.0.16	libftd3xx_32.dylib
	ARM		1.0.16	libftd3xx_ARM.dylib

Table 1.1.0 – D3XX Dynamic Library OS Associations

Figure 1.1.0 shows how PyD3XX sits in your Windows application's hierarchy. If PyD3XX detects the WinUSB driver for FTDI's FT60X devices, it will load the WinUSB version of the D3XX library.

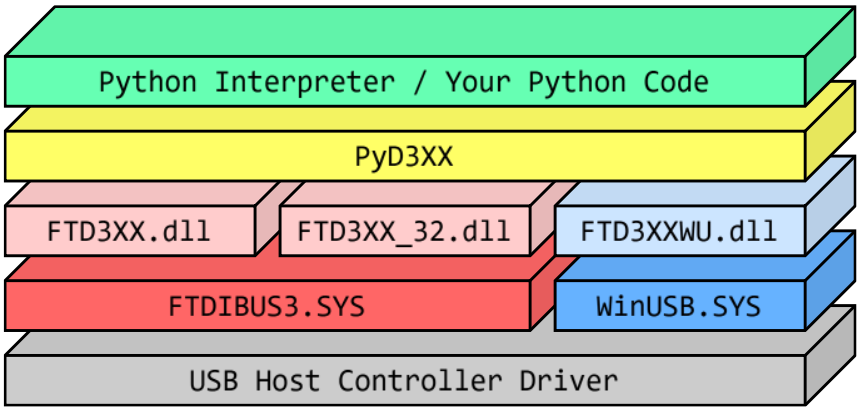

Figure 1.1.0 – PyD3XX Windows Hierarchy

1.2 – Interfaces, Channels, & Endpoints

FTDI's FT60X devices come with two interfaces, a communication interface and a data interface. The data interface can have up to 4 channels when the device is configured in FT600 mode. Each channel comes with a read and write endpoint, adding up to 8 total data endpoints for the data interface. The communication interface comes with two endpoints, a OUT BULK endpoint for receiving Session List commands from the host, and a IN INTERRUPT endpoint for Notification List commands.

Interface	FIFO Index	Pipe Index	Channel	Endpoint/s	Description
0	N/A	0	N/A	0x01	OUT BULK endpoint for Session List commands.
		1		0x81	IN INTERRUPT endpoint for Notification List commands.
1	0-3	0,2,4,6	1-4	0x02-0x05	OUT BULK endpoint for application write access. Writes data out to channels.
		1,3,5,7		0x82-0x85	IN BULK endpoint for application read access. Reads data in from channels.

Table 1.2.0 – FT60X Interfaces, Channels, & Endpoints

1.3 – SetPrintLevel()

D3XX C API Equivalent = NONE. PyD3XX exclusive.

Summary

Sets the PrintLevel behavior of PyD3XX. By default, PrintLevel is set to PRINT_NONE, where PyD3XX will not print out any messages.

Parameters

Type	Name	Description		
int	PrintLevel	A bitmask controlling what messages PyD3XX will print.		
		PRINT_NONE	0b00000	Print no messages.
		PRINT_ERROR_CRITICAL	0b00001	Print critical error messages.
		PRINT_ERROR_MAJOR	0b00010	Print major error messages.
		PRINT_ERROR_MINOR	0b00100	Print minor error messages.
		PRINT_ERROR_ALL	0b00111	Print all error messages.
		PRINT_INFO_START	0b01000	Print informational startup messages.
		PRINT_INFO_DEVICE	0b10000	Print device information.
		PRINT_INFO_ALL	0b11000	Print all informational messages.
		PRINT_ALL	0b11111	Print all messages.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code	`PyD3XX.SetPrintLevel(PyD3XX.PRINT_ALL) # Make PyD3XX print everything.`		
State	Successfully loaded dynamic library file.	Output	PyD3XX - Startup INFO: DID NOT DETECT WinUSB: Will use D3XX dynamic library. PyD3XX - Startup INFO: DETECTED 64-BIT PYTHON ENVIRONMENT: LOADING 64-bit dynamic library file & setting 64-bit argtypes. PyD3XX - Startup INFO: Successfully loaded FTDI D3XX dynamic library.

2 – PyD3XX Functions

This section goes over the available PyD3XX functions which can vary based on the operating system your Python code is running on. The "Platform" variable in PyD3XX is populated with one of the strings in Table 2.0.0 based off the operating system PyD3XX detected.

Operating System	Platform
Windows	windows
Linux	linux
Unknown OS	
macOS	darwin

Table 2.0.0 – Platform String Values

2.1 – FT_CreateDeviceInfoList()

D3XX C API Equivalent = FT_CreateDeviceInfoList()

Summary

Builds an internal device information list of all D3XX devices connected to the system and returns the total number of devices. The internal device information list does not change even if devices are added/removed from the system until FT_CreateDeviceInfoList() is called again.

Parameters

None

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	DeviceCount	Number of D3XX devices connected to the system.

Example Code

Example Code

```
Status, DeviceCount = PyD3XX.FT_CreateDeviceInfoList() # Create a device info list.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CREATE DEVICE INFO LIST: ABORTING")
    exit()
print(str(DeviceCount) + " Devices detected.")
if (DeviceCount == 0):
    print("NO DEVICES DETECTED: ABORTING")
    exit()
```

State	Output
One device.	1 Devices detected.
No devices.	0 Devices detected. NO DEVICES DETECTED: ABORTING

2.2 – FT_GetDeviceInfoList()

D3XX C API Equivalent = FT_GetDeviceInfoList()

Summary

Returns a list of FT_Device objects and the number of D3XX devices in the list.

This function should only be called after calling FT_CreateDeviceInfoList() which creates the list this function retrieves its device information from.

Parameters

Type	Name	Description
int	DeviceCount	The number of devices you want to retrieve from the internal device information list.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
list[FT_Device]	Devices	A list of FT_Device objects.

Example Code

```
Status, Devices = PyD3XX.FT_GetDeviceInfoList(DeviceCount) # This will create a list.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CREATE DEVICE INFO LIST: ABORTING")
    exit()
for i in range(0, DeviceCount): # Get device info at each index from 0->(DeviceCount - 1).
    if(Devices[i].Flags & PyD3XX.FT_FLAGS_OPENED):
        print("Device " + str(i) + " is opened by another process!")
        continue # Device info not valid. Move onto next device.
    print("---| Device " + str(i) + " |---")
```

State	Output
2 unopened devices.	---| Device 0 |--- ---| Device 1 |---
1 open, 1 unopened.	---| Device 0 |--- Device 1 is opened by another process!

	2 opened devices.		Device 0 is opened by another process! Device 1 is opened by another process!

2.2B – FT_GetDeviceInfoDict()

D3XX C API Equivalent = NONE. PyD3XX exclusive.

Summary

Returns a dictionary of FT_Device objects and the number of D3XX devices in the dictionary. This is meant as an alternative to FT_GetDeviceInfoList() in the scenario where you want a Python dictionary instead of a Python list of FT_Device objects.

This function should only be called after calling FT_CreateDeviceInfoList() which creates the list this function retrieves its device information from.

Parameters

Type	Name	Description
int	DeviceCount	The number of devices you want to retrieve from the internal device information list.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
dict[int, FT_Device]	Devices	A dictionary of FT_Device objects.

Example Code

Example Code

```
Status, Devices = PyD3XX.FT_GetDeviceInfoDict(DeviceCount) # This will create a Python
dictionary.
    if Status != PyD3XX.FT_OK:
        print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CREATE DEVICE INFO LIST:
ABORTING")
        exit()
for i in range(0, DeviceCount): # Get device info at each index from 0->(DeviceCount - 1).
    if(Devices[i].Flags & PyD3XX.FT_FLAGS_OPENED):
        print("Device " + str(i) + " is opened by another process!")
        continue # Device info not valid. Move onto next device.
    print("---| Device " + str(i) + " |---")
```

State	Output
2 unopened devices.	---| Device 0 |--- ---| Device 1 |---
1 open, 1 unopened.	---| Device 0 |--- Device 1 is opened by another process!
2 opened devices.	Device 0 is opened by another process! Device 1 is opened by another process!

2.3 – FT_GetDeviceInfoDetail()

D3XX C API Equivalent = FT_GetDeviceInfoDetail()

Summary

Returns an FT_Device object that contains all of the information of the device recorded at the given index of the internal device information list. This does not create/open a device.

This function should only be called after calling FT_CreateDeviceInfoList() which creates the list this function retrieves its device information from.

Parameters

Type	Name	Description
int	Index	Index of the device entry to retrieve from the internal device information list.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_Device	ReturnDevice	The FT_Device object with the retrieved device information.

Example Code

Example Code

```
Status, Device = PyD3XX.FT_GetDeviceInfoDetail(0) # Get info of a device at index 0.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO GET INFO FOR DEVICE 0")
else:
    if(Device.Flags & PyD3XX.FT_FLAGS_OPENED):
        print("Device 0 is opened by another process!")
    else:
        if(Device.Type == PyD3XX.FT_DEVICE_601):
                print("Device 0 is a FT601 device!")
        elif(Device.Type == PyD3XX.FT_DEVICE_600):
            print("Device 0 is a FT600 device!")
```

State	Output
FT600 connected.	Device 0 is a FT600 device!
FT601 connected.	Device 0 is a FT601 device!
FT60X open in another program.	Device 0 is opened by another process!

2.4 – FT_ListDevices()

D3XX C API Equivalent = FT_ListDevices()

Summary

Can return the total number of devices, the serial number or description of a single device, or return the serial number or description of multiple devices. This function will always return the Status integer. This function will also return either an integer, string, or list of strings.

Parameters

Type	Name	Description
int	IndexCount	If (Flags & FT_LIST_BY_INDEX) != 0, then IndexCount is the index of the device whose serial number or description you want returned. If (Flags & FT_LIST_ALL) != 0, then IndexCount is the number of devices you

		want to get the info of. This should never be more than the number of devices connected to the system.
int	Flags	A bit field that controls what data you're requesting from FT_ListDevices. FT_LIST_NUMBER_ONLY = Return the number of devices connected to the system. FT_LIST_BY_INDEX = Return the serial number or description of the device at index IndexCount. FT_LIST_ALL = Return a list of the serial number or description of IndexCount devices. FT_OPEN_BY_SERIAL_NUMBER = Return serial number/s. FT_OPEN_BY_DESCRIPTION = Return description/s.

Return Values

Type	Name	Description
int	Status	FT_OK if successful. Status will always be returned.
Condition: (Flags & FT_LIST_NUMBER_ONLY) != 0		
int	DeviceCount	The number of devices currently connected to the system.
Condition: (Flags & FT_LIST_BY_INDEX) != 0		

str	SerialDesc	If (Flags & FT_OPEN_BY_SERIAL_NUMBER) != 0, then it is the serial number of the device at the given index.
		If (Flags & FT_OPEN_BY_DESCRIPTION) != 0, then it is the description of the device at the given index.
		If there was an error, then it is FT_STATUS_STR[Status].

Condition: (Flags & FT_LIST_ALL) != 0		

list[str]	SeriDescList	If (Flags & FT_OPEN_BY_SERIAL_NUMBER) != 0, then it is list of IndexCount device's serial numbers.
		If (Flags & FT_OPEN_BY_DESCRIPTION) != 0, then it is list of IndexCount device's descriptions.
		If there was an error, then it is a list of IndexCount FT_STATUS_STR[Status].

Example Code

Example Code

```
Status, Count = PyD3XX.FT_ListDevices(PyD3XX.NULL, PyD3XX.FT_LIST_NUMBER_ONLY)
print("Device Count: " + str(Count))
Status, SerialNumber = PyD3XX.FT_ListDevices(0, PyD3XX.FT_LIST_BY_INDEX |
PyD3XX.FT_OPEN_BY_SERIAL_NUMBER)
print("Device 0 Serial Number: " + str(SerialNumber))
Status, Description = PyD3XX.FT_ListDevices(0, PyD3XX.FT_LIST_BY_INDEX |
PyD3XX.FT_OPEN_BY_DESCRIPTION)
print("Device 0 Description: " + str(Description))
Status, SerialNumbers = PyD3XX.FT_ListDevices(Count, PyD3XX.FT_LIST_ALL |
PyD3XX.FT_OPEN_BY_SERIAL_NUMBER)
for i in range(Count):
    print("Device " + str(i) + " Serial Number: " + str(SerialNumbers[i]))
Status, Descriptions = PyD3XX.FT_ListDevices(Count, PyD3XX.FT_LIST_ALL |
PyD3XX.FT_OPEN_BY_DESCRIPTION)
for i in range(Count):
    print("Device " + str(i) + " Description: " + str(Descriptions[i]))
```

State	Two devices connected.	Output	Device Count: 2 Device 0 Serial Number: 000000000001 Device 0 Description: FTDI SuperSpeed-FIFO Bridge Device 0 Serial Number: 000000000001 Device 1 Serial Number: 44TXERgJa3h2IHD Device 0 Description: FTDI SuperSpeed-FIFO Bridge Device 1 Description: TestDevice

2.5 – FT_Create()

D3XX C API Equivalent = FT_Create()

Summary

Opens the given device by index, serial number, or description for sole use by your program. You must give this function an FT_Device object created by FT_GetDeviceInfoList(), FT_GetDeviceInfoDict(), or FT_GetDeviceInfoDetail().

Parameters

Type	Name	Description
int, str	Identifier	The index, serial number, or description of the device you want to open.
int	OpenFlag	A flag that controls whether the function opens the device by index, serial number, or description.
FT_Device	Device	The device you want to open.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status, Device = PyD3XX.FT_GetDeviceInfoDetail(0) # Get info of a device at index 0.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO GET INFO FOR DEVICE 0")
print("Device (index, serial, desc): 0, " + Device.SerialNumber + ", " + Device.Description);
Status = PyD3XX.FT_Create(0, PyD3XX.FT_OPEN_BY_INDEX, Device)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO OPEN DEVICE: ABORTING")
    exit()
PyD3XX.FT_Close(Device)
print("Successfully opened and closed device by index!");
Status = PyD3XX.FT_Create("44TXERgJa3h2IHD", PyD3XX.FT_OPEN_BY_SERIAL_NUMBER, Device)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO OPEN DEVICE: ABORTING")
    exit()
PyD3XX.FT_Close(Device)
print("Successfully opened and closed device by serial number!");
Status = PyD3XX.FT_Create("TestDevice", PyD3XX.FT_OPEN_BY_DESCRIPTION, Device)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO OPEN DEVICE: ABORTING")
    exit()
PyD3XX.FT_Close(Device)
print("Successfully opened and closed device by description!");
```

State	Output
Device not in use by another program.	Device (index, serial, desc): 0, 44TXERgJa3h2IHD, TestDevice Successfully opened and closed device by index! Successfully opened and closed device by serial number! Successfully opened and closed device by description!

2.6 – FT_Close()

D3XX C API Equivalent = FT_Close()

Summary

Closes a device opened by FT_Create().

Parameters

Type	Name	Description
FT_Device	Device	The device you want to close.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_Create(0, PyD3XX.FT_OPEN_BY_INDEX, Device) # Open the device.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO OPEN DEVICE: ABORTING")
    exit()
PyD3XX.FT_Close(Device)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CLOSE DEVICE: ABORTING")
    exit()
print("Successfully opened and closed device by index!");
```

State	Output
Device connected.	Successfully opened and closed device by index!
No devices.	FT_DEVICE_NOT_FOUND | FAILED TO OPEN DEVICE: ABORTING

2.7 – FT_WritePipe()

D3XX C API Equivalent = FT_WritePipe()

Summary

Write data from an FT_Buffer object to a given pipe.

Parameters

Type	Name	Description	
FT_Device	Device	The device whose pipe you want to write to.	
FT_Pipe | int	Pipe_Endpoint	Windows	An FT_Pipe object of the pipe you want to write to.
		Linux MacOS	An integer of the endpoint you want to write to. Values 0x82-0x85 = Channels 1-4
FT_Buffer	Buffer	The buffer whose data you'll write to the pipe.	
int	BufferLength	The amount of data from the buffer that you want to write.	
FT_Overlapped | int	Overlapped_TimeoutMs	Windows	An FT_Overlapped structure if doing an asynchronous write or NULL for a synchronous write.
		Linux MacOS	An int for the number of milliseconds FT_WritePipe can hang on before timing out.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	BytesTransferred	The number of bytes successfully written to the pipe.

Example Code

Example Code

```
Pipes = {}
for i in range(2): # Get in and out pipes for channel 1.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i)
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, int("0x82", 16), 1, 0)
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, Pipes[1], 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
    print("Channel 1 = 0x" + ReadValue)
else:
    print(PyD3XX.FT_STATUS_STR[Status] + " | Failed to read data!: ABORTING")
    exit()
WriteBuffer = PyD3XX.FT_Buffer.from_int(int.from_bytes(ReadBuffer.Value(), "little") + 1)
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, BytesWrote = PyD3XX.FT_WritePipe(Device, int("0x02", 16), WriteBuffer, 1, 0)
else:
    Status, BytesWrote = PyD3XX.FT_WritePipe(Device, Pipes[0], WriteBuffer, 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    WriteValue = format(int.from_bytes(WriteBuffer.Value(), "little"), "x").zfill(1*2)
    print("Wrote 0x" + WriteValue + " to Channel 1!")
else:
    print(PyD3XX.FT_STATUS_STR[Status] + " | Failed to write data!: ABORTING")
    exit()
```

State	Output
First success.	Channel 1 = 0x01 Wrote 0x02 to Channel 1!
Second success.	Channel 1 = 0x02 Wrote 0x03 to Channel 1!

2.8 – FT_ReadPipe()

D3XX C API Equivalent = FT_ReadPipe()

Summary

Returns an FT_Buffer object with data read from a given pipe.

Parameters

Type	Name	Description
FT_Device	Device	The device whose pipe you want to read from.

FT_Pipe | int	Pipe_Endpoint	Windows	An FT_Pipe object of the pipe you want to read from.
		Linux MacOS	An integer of the endpoint you want to read from. Values 0x02-0x05 = Channels 1-4
int	BufferLength	The amount of data you want to read from the pipe.	
FT_Overlapped | int	Overlapped_TimeoutMs	Windows	An FT_Overlapped structure if doing an asynchronous read or NULL for a synchronous read.
		Linux MacOS	An int for the number of milliseconds FT_ReadPipe can hang on before timing out.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_Buffer	Buffer	The buffer with data read from the pipe.
int	BytesTransferred	The number of bytes successfully read from the pipe into the buffer.

Example Code

Example Code

```
Pipes = {}
for i in range(2): # Get in and out pipes for channel 1.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i)
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, int("0x82", 16), 1, 0)
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, Pipes[1], 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
    print("Channel 1 = 0x" + ReadValue)
else:
    print(PyD3XX.FT_STATUS_STR[Status] + " | Failed to read data!: ABORTING")
    exit()
WriteBuffer = PyD3XX.FT_Buffer.from_int(int.from_bytes(ReadBuffer.Value(), "little") + 1)
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, BytesWrote = PyD3XX.FT_WritePipe(Device, int("0x02", 16), WriteBuffer, 1, 0)
else:
    Status, BytesWrote = PyD3XX.FT_WritePipe(Device, Pipes[0], WriteBuffer, 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    WriteValue = format(int.from_bytes(WriteBuffer.Value(), "little"), "x").zfill(1*2)
    print("Wrote 0x" + WriteValue + " to Channel 1!")
else:
    print(PyD3XX.FT_STATUS_STR[Status] + " | Failed to write data!: ABORTING")
    exit()
```

State	Output
First success.	Channel 1 = 0x01 Wrote 0x02 to Channel 1!
Second success.	Channel 1 = 0x02 Wrote 0x03 to Channel 1!

2.9 – FT_WritePipeEx()

D3XX C API Equivalent = FT_WritePipeEx()

Summary

Write data from an FT_Buffer object to a given pipe. FT_WritePipeEx() has lower latency than FT_WritePipe() for asynchronous transfers when used along with FT_SetStreamPipe.

Note: FT_WritePipeEx() supports a maximum buffer size of 1 MiB. This is to guarantee lower latencies.

Parameters

Type	Name	Description

FT_Device	Device	The device whose pipe you want to write to.	
FT_Pipe | int	Pipe_FIFOindex	Windows	An FT_Pipe object of the pipe you want to write to.
		Linux MacOS	An integer FIFO index of the channel you want to write to. Values 0-3 = Channels 1-4
FT_Buffer	Buffer	The buffer whose data you'll write to the pipe.	
int	BufferLength	The amount of data from the buffer that you want to write.	
FT_Overlapped | int	Overlapped_TimeoutMs	Windows	An FT_Overlapped structure if doing an asynchronous write or NULL for a synchronous write.
		Linux MacOS	An int for the number of milliseconds FT_WritePipeEx can hang on before timing out.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	BytesTransferred	The number of bytes successfully written to the pipe.

Example Code

Example Code

```
Pipes = {}
for i in range(2): # Get in and out pipes for channel 1.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i)
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeEx(Device, 0, 1, 0)
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeEx(Device, Pipes[1], 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
    print("Channel 1 = 0x" + ReadValue)
else:
    print("Failed to read data!: ABORTING")
    exit()
WriteBuffer = PyD3XX.FT_Buffer.from_int(int.from_bytes(ReadBuffer.Value(), "little") + 1)
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, BytesWrote = PyD3XX.FT_WritePipeEx(Device, 0, WriteBuffer, 1, 0)
else:
    Status, BytesWrote = PyD3XX.FT_WritePipeEx(Device, Pipes[0], WriteBuffer, 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    WriteValue = format(int.from_bytes(WriteBuffer.Value(), "little"), "x").zfill(1*2)
    print("Wrote 0x" + WriteValue + " to Channel 1!")
else:
    print("Failed to write data!: ABORTING")
    exit()
```

State	Output
First success.	Channel 1 = 0x01 Wrote 0x02 to Channel 1!
Second success.	Channel 1 = 0x02 Wrote 0x03 to Channel 1!

2.9B – FT_WritePipeAsync()

D3XX C API Equivalent = FT_WritePipeAsync()
WARNING – This function is exclusive to Linux/MacOS systems.

Summary

Asynchronously write data from an FT_Buffer object to a given pipe.

Parameters

Type	Name	Description
FT_Device	Device	The device whose pipe you want to write to.
int	FIFOindex	An integer FIFO index of the channel you want to write to. Values 0-3 = Channels 1-4

FT_Buffer	Buffer	The buffer whose data you'll write to the pipe.
int	BufferLength	The amount of data from the buffer that you want to write.
FT_Overlapped	Overlapped	An FT_Overlapped structure for doing an asynchronous write.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	BytesTransferred	The number of bytes successfully written to the pipe.

Example Code

Example Code

```
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeAsync(Device, 0, 1, Overlaps[1])
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, Pipes[1], 1, Overlaps[1])
Attempts = 0
Status = PyD3XX.FT_IO_INCOMPLETE
while(Status == PyD3XX.FT_IO_INCOMPLETE):
    Status, BytesTransferred = PyD3XX.FT_GetOverlappedResult(Device, Overlaps[1], False)
    print("Waiting on input...")
    Attempts += 1
    time.sleep(1)
    if(Status == PyD3XX.FT_OK): break
    if(Attempts == 5):
        print(PyD3XX.FT_STATUS_STR[Status] + " | ERROR: Async read failed!")
        exit()
print(PyD3XX.FT_STATUS_STR[Status] + " | FT_ReadPipe was successful.")
ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
print("Channel 1 = 0x" + ReadValue)

WriteBuffer = PyD3XX.FT_Buffer.from_int(int.from_bytes(ReadBuffer.Value(), "little") + 1)
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, BytesWrote = PyD3XX.FT_WritePipeAsync(Device, 0, WriteBuffer, 1, Overlaps[0])
else:
    Status, BytesWrote = PyD3XX.FT_WritePipe(Device, Pipes[0], WriteBuffer, 1, Overlaps[0])
Attempts = 0
Status = PyD3XX.FT_IO_INCOMPLETE
while(Status == PyD3XX.FT_IO_INCOMPLETE):
    Status, BytesTransferred = PyD3XX.FT_GetOverlappedResult(Device, Overlaps[0], False)
    print("Waiting on output...")
    Attempts += 1
    time.sleep(1)
    if(Status == PyD3XX.FT_OK): break
    if(Attempts == 5):
        print(PyD3XX.FT_STATUS_STR[Status] + " | ERROR: Async write failed!")
        exit()
print(PyD3XX.FT_STATUS_STR[Status] + " | FT_WritePipe was successful.")
for i in range(2): # Release overlaps since we're done using them.
    Status = PyD3XX.FT_ReleaseOverlapped(Device, Overlaps[i])
    if Status != PyD3XX.FT_OK:
        print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO RELEASE OVERLAP " +
str(i))
```

State	Output
First success.	Waiting on input... FT_OK | FT_ReadPipe was successful. Channel 1 = 0x01 Waiting on output... Waiting on output... FT_OK | FT_WritePipe was successful.
Second success.	Waiting on input... Waiting on input... FT_OK | FT_ReadPipe was successful. Channel 1 = 0x02 Waiting on output... Waiting on output... FT_OK | FT_WritePipe was successful.

2.10 – FT_ReadPipeEx()

D3XX C API Equivalent = FT_ReadPipeEx()

Summary

Returns an FT_Buffer object with data read from a given pipe. FT_ReadPipeEx() has lower latency than FT_ReadPipe() for asynchronous transfers when used along with FT_SetStreamPipe.

Parameters

Type	Name		Description
FT_Device	Device		The device whose pipe you want to read from.
FT_Pipe | int	Pipe_FIFOindex	Windows	An FT_Pipe object of the pipe you want to read from.
		Linux MacOS	An integer FIFO index of the channel you want to read from. Values 0-3 = Channels 1-4
int	BufferLength		The amount of data you want to read from the pipe.
FT_Overlapped | int	Overlapped_TimeoutMs	Windows	An FT_Overlapped structure if doing an asynchronous read or NULL for a synchronous read.
		Linux MacOS	An int for the number of milliseconds FT_ReadPipeEx can hang on before timing out.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_Buffer	Buffer	The buffer with data read from the pipe.
int	BytesTransferred	The number of bytes successfully read from the pipe into the buffer.

Example Code

Example Code

```
Pipes = {}
for i in range(2): # Get in and out pipes for channel 1.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i)
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeEx(Device, 0, 1, 0)
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeEx(Device, Pipes[1], 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
    print("Channel 1 = 0x" + ReadValue)
else:
    print("Failed to read data!: ABORTING")
    exit()
WriteBuffer = PyD3XX.FT_Buffer.from_int(int.from_bytes(ReadBuffer.Value(), "little") + 1)
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, BytesWrote = PyD3XX.FT_WritePipeEx(Device, 0, WriteBuffer, 1, 0)
else:
    Status, BytesWrote = PyD3XX.FT_WritePipeEx(Device, Pipes[0], WriteBuffer, 1, PyD3XX.NULL)
if(Status == PyD3XX.FT_OK):
    WriteValue = format(int.from_bytes(WriteBuffer.Value(), "little"), "x").zfill(1*2)
    print("Wrote 0x" + WriteValue + " to Channel 1!")
else:
    print("Failed to write data!: ABORTING")
    exit()
```

State	Output
First success.	Channel 1 = 0x01 Wrote 0x02 to Channel 1!
Second success.	Channel 1 = 0x02 Wrote 0x03 to Channel 1!

2.10B – FT_ReadPipeAsync()

D3XX C API Equivalent = FT_ReadPipeAsync()

Summary

Returns an FT_Buffer object with data asynchronously read from a given pipe.

Parameters

Type	Name	Description
FT_Device	Device	The device whose pipe you want to read from.
int	FIFOindex	An integer FIFO index of the channel you want to read from. Values 0-3 = Channels 1-4

int	BufferLength	The amount of data you want to read from the pipe.
FT_Overlapped	Overlapped	An FT_Overlapped structure for doing an asynchronous read.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_Buffer	Buffer	The buffer with data read from the pipe.
int	BytesTransferred	The number of bytes successfully read from the pipe into the buffer.

Example Code

Example Code

```
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeAsync(Device, 0, 1, Overlaps[1])
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, Pipes[1], 1, Overlaps[1])
Attempts = 0
Status = PyD3XX.FT_IO_INCOMPLETE
while(Status == PyD3XX.FT_IO_INCOMPLETE):
    Status, BytesTransferred = PyD3XX.FT_GetOverlappedResult(Device, Overlaps[1], False)
    print("Waiting on input...")
    Attempts += 1
    time.sleep(1)
    if(Status == PyD3XX.FT_OK): break
    if(Attempts == 5):
        print(PyD3XX.FT_STATUS_STR[Status] + " | ERROR: Async read failed!")
        exit()
print(PyD3XX.FT_STATUS_STR[Status] + " | FT_ReadPipe was successful.")
ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
print("Channel 1 = 0x" + ReadValue)

WriteBuffer = PyD3XX.FT_Buffer.from_int(int.from_bytes(ReadBuffer.Value(), "little") + 1)
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, BytesWrote = PyD3XX.FT_WritePipeAsync(Device, 0, WriteBuffer, 1, Overlaps[0])
else:
    Status, BytesWrote = PyD3XX.FT_WritePipe(Device, Pipes[0], WriteBuffer, 1, Overlaps[0])
Attempts = 0
Status = PyD3XX.FT_IO_INCOMPLETE
while(Status == PyD3XX.FT_IO_INCOMPLETE):
    Status, BytesTransferred = PyD3XX.FT_GetOverlappedResult(Device, Overlaps[0], False)
    print("Waiting on output...")
    Attempts += 1
    time.sleep(1)
    if(Status == PyD3XX.FT_OK): break
    if(Attempts == 5):
        print(PyD3XX.FT_STATUS_STR[Status] + " | ERROR: Async write failed!")
        exit()
print(PyD3XX.FT_STATUS_STR[Status] + " | FT_WritePipe was successful.")
for i in range(2): # Release overlaps since we're done using them.
    Status = PyD3XX.FT_ReleaseOverlapped(Device, Overlaps[i])
    if Status != PyD3XX.FT_OK:
        print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO RELEASE OVERLAP " +
str(i))
```

State	First success.	Output	Waiting on input... FT_OK | FT_ReadPipe was successful. Channel 1 = 0x01 Waiting on output... Waiting on output... FT_OK | FT_WritePipe was successful.
	Second success.		Waiting on input... Waiting on input... FT_OK | FT_ReadPipe was successful. Channel 1 = 0x02 Waiting on output... Waiting on output... FT_OK | FT_WritePipe was successful.

2.11 – FT_GetOverlappedResult()

D3XX C API Equivalent = FT_GetOverlappedResult()

Summary

Returns the result of an overlapped operation to a pipe.

Parameters

Type	Name	Description
FT_Device	Device	The device that had an overlapped operation done.
FT_Overlapped	Overlap	The overlapped structure that was used in a read/write operation.
bool	Wait	True: FT_GetOverlappedResult() will hang until the read/write operation has completed. False: FT_GetOverlappedResult() will immediately return whether the read/write operation has completed or not.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	LengthTransferred	The number of bytes successfully transferred in the read/write operation.

Example Code

Example Code

```
Pipes = {}
Overlaps = {}
for i in range(2): # Get in and out pipes for channel 1 and create overlaps.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i)
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
    Status, Overlaps[i] = PyD3XX.FT_InitializeOverlapped(Device)
    if(Status != PyD3XX.FT_OK):
        print("FAILED TO CREATE OVERLAP FOR PIPE " + str(i) + ": ABORTING")
        exit()
    Status = PyD3XX.FT_SetPipeTimeout(Device, Pipes[i], 100)
    if(Status != PyD3XX.FT_OK):
        print("WARNING: FAILED TO SET PIPE TIMEOUT TO 100")
        print("Status = " + PyD3XX.FT_STATUS_STR[Status])
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeAsync(Device, 0, 1, Overlaps[1])
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, Pipes[1], 1, Overlaps[1])
Attempts = 0
Status = PyD3XX.FT_IO_INCOMPLETE
while(Status == PyD3XX.FT_IO_INCOMPLETE):
    Status, BytesTransferred = PyD3XX.FT_GetOverlappedResult(Device, Overlaps[1], False)
    print("Waiting on input...")
    Attempts += 1
    time.sleep(1)
    if((Status != PyD3XX.FT_OK) and (Status != PyD3XX.FT_IO_PENDING)) or (Attempts == 3):
        StatusAP = PyD3XX.FT_AbortPipe(Device, Pipes[1])
        if(StatusAP != PyD3XX.FT_OK):
            print("FT_AbortPipe failed!")
        else:
            print("Aborted read pipe!")
        break
print(PyD3XX.FT_STATUS_STR[Status] + " | Encountered error or FT_ReadPipe was successful.")
ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
print("Channel 1 = 0x" + ReadValue)
for i in range(2): # Release overlaps since we're done using them.
    Status = PyD3XX.FT_ReleaseOverlapped(Device, Overlaps[i])
    if Status != PyD3XX.FT_OK:
        print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO RELEASE OVERLAP " +
str(i))
```

State	Output
Successfully read one byte.	Waiting on input... FT_OK | Encountered error or FT_ReadPipe was successful. Channel 1 = 0x01
Overlapped read operation did not complete.	Waiting on input... Aborted read pipe! FT_IO_INCOMPLETE | Encountered error or FT_ReadPipe was successful. Channel 1 = 0x00

2.12 – FT_InitializeOverlapped()

D3XX C API Equivalent = FT_InitializeOverlapped()

Summary

Create an FT_Overlapped object to be used asynchronously with the FT_WritePipe and FT_ReadPipe functions. The FT_Overlapped object should be released with FT_ReleaseOverlapped after it is no longer needed.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to create an FT_Overlapped object from.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_Overlapped	Overlap	The overlapped structure necessary for asynchronous read and write operations. Release with FT_ReleaseOverlapped when no longer needed.

Example Code

Example Code:

```
Status, Overlap = PyD3XX.FT_InitializeOverlapped(Device)
if(Status != PyD3XX.FT_OK):
    print("FAILED TO CREATE OVERLAP: ABORTING")
    exit()
print("Successfully created overlap!")
```

State	Output
Success	Successfully created overlap!

2.13 – FT_ReleaseOverlapped()

D3XX C API Equivalent = FT_ReleaseOverlapped()

Summary

Releases the resources used by an FT_Overlapped object. Call this when an FT_Overlapped object is no longer needed.

Parameters

Type	Name	Description

FT_Device	Device	The device you want to release an overlap from.
FT_Overlapped	Overlap	The overlap to be released.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status, Overlap = PyD3XX.FT_InitializeOverlapped(Device)
if(Status != PyD3XX.FT_OK):
    print("FAILED TO CREATE OVERLAP: ABORTING")
    exit()
print("Successfully created overlap!")
Status = PyD3XX.FT_ReleaseOverlapped(Device, Overlap)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO RELEASE OVERLAP.")
else:
    print("Successfully released overlap!")
```

State	Output
Success	Successfully created overlap! Successfully released overlap!

2.14 – FT_SetStreamPipe()

D3XX C API Equivalent = FT_SetStreamPipe()

Summary

Set the streaming protocol for pipes with a fixed transfer size in bytes.

Note: For the WinUSB drivers, the StreamSize parameter must be a multiple of the max packet size. Otherwise, future read calls will fail.
SuperSpeed: MaxPacketSize = 1024 bytes.
Hi-Speed: MaxPacketSize = 512 bytes.
Full Speed: MaxPacketSize = 64 bytes.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to set a streaming protocol for.
bool	AllWritePipes	True: Set the streaming protocol for all write pipes. False: Ignore
bool	AllReadPipes	True: Set the streaming protocol for all read pipes. False: Ignore
FT_Pipe	Pipe	If AllWritePipes & AllReadPipes are both False, then this specific pipe's streaming protocol is set. Otherwise this parameter is ignored and you should pass NULL instead.
int	StreamSize	Transfer size in bytes used for read/write operations.

Return Values

Type	Name	Description

int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_SetStreamPipe(Device, True, True, PyD3XX.NULL, 1024)
if Status != PyD3XX.FT_OK:
    print("FAILED TO SET STREAM SIZE FOR ALL PIPES.")
    exit()
else:
    print("Stream size for all pipes set to 1024 bytes!")
```

State	Output
Success	Stream size for all pipes set to 1024 bytes!

2.15 – FT_ClearStreamPipe()

D3XX C API Equivalent = FT_ClearStreamPipe()

Summary

Clear the previously set streaming protocol transfer for pipes.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to clear a streaming protocol for.
bool	AllWritePipes	True: Clear the streaming protocol for all write pipes. False: Ignore
bool	AllReadPipes	True: Clear the streaming protocol for all read pipes. False: Ignore
FT_Pipe	Pipe	If AllWritePipes & AllReadPipes are both False, then this specific pipe's streaming protocol is cleared. Otherwise this parameter is ignored and you should pass NULL instead.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Pipes = {}
for i in range(2): # Get in and out pipes for channel 1.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i)
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
Status = PyD3XX.FT_SetStreamPipe(Device, True, True, PyD3XX.NULL, 1)
if Status != PyD3XX.FT_OK:
    print("FAILED TO SET STREAM SIZE FOR ALL PIPES.")
    exit()
else:
    print("Stream size for all pipes set to 1 byte!")
Status = PyD3XX.FT_ClearStreamPipe(Device, False, False, Pipes[1])
if Status != PyD3XX.FT_OK:
    print("FAILED TO CLEAR STREAM SIZE FOR CH1 READ PIPE.")
    exit()
else:
    print("Cleared stream size for CH1 read pipe!")
```

State	Output
Success	Stream size for all pipes set to 1 byte! Cleared stream size for CH1 read pipe!

2.16 – FT_SetPipeTimeout()

D3XX C API Equivalent = FT_SetPipeTimeout()

Summary

Sets the timeout value for a given FT_Pipe object (endpoint). FT_ReadPipe and FT_WritePipe will timeout if they hang for however many seconds the timeout value is set to. If the FT_Device is closed and re-opened, then the pipe timeout value will reset to the driver default.

Parameters

Type	Name	Description
FT_Device	Device	The device whose FT_Pipe you want to change the timeout value of.
FT_Pipe	Pipe	The pipe (endpoint) you want to change the timeout value of.
int	Timeout	The new timeout value you want to set for the pipe (endpoint).

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Pipes = {}
for i in range(CHANNEL_COUNT * 2): # Set timeout of all pipes to 100.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i) # Get pipe information.
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
    Status_SPT = PyD3XX.FT_SetPipeTimeout(Device, Pipes[i], 100)
    if(PyD3XX.Platform == "windows"): # FT_GetPipeTimeout not in D3XX Linux library.
        Status, TimeOut = PyD3XX.FT_GetPipeTimeout(Device, Pipes[i])
        if (Status != PyD3XX.FT_OK) or (TimeOut != 100):
            print("FAILED TO SET AND/OR CONFIRM PIPE TIMEOUT TO 100: ABORTING")
            print("Status = " + PyD3XX.FT_STATUS_STR[Status] + " | Timeout = " +
str(TimeOut))
            exit()
    if(Status_SPT != PyD3XX.FT_OK):
        print("FAILED TO SET AND/OR CONFIRM PIPE TIMEOUT TO 100: ABORTING")
        print("Status = " + PyD3XX.FT_STATUS_STR[Status] + " | Timeout = " + str(TimeOut))
    else:
        print("Successfully set timeout of pipe " + str(i) + " to 100!")
```

State	Output
CHANNEL_COUNT = 1 and success.	Successfully set timeout of pipe 0 to 100! Successfully set timeout of pipe 1 to 100!

2.17 – FT_GetPipeTimeout()

D3XX C API Equivalent = FT_GetPipeTimeout()
WARNING – This function is exclusive to Windows systems.

Summary

Gets the timeout value for a given FT_Pipe object (IN endpoint). This function is exclusive to Windows systems.

Parameters

Type	Name	Description
FT_Device	Device	The device whose FT_Pipe you want to get the timeout value of.
FT_Pipe	Pipe	The pipe (endpoint) you want to get the timeout value of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	Timeout	The timeout value of the pipe (IN endpoint).

Example Code

Example Code

```
Pipes = {}
for i in range(CHANNEL_COUNT * 2): # Set timeout of all pipes to 100.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i) # Get pipe information.
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
    Status_SPT = PyD3XX.FT_SetPipeTimeout(Device, Pipes[i], 100)
    if(PyD3XX.Platform == "windows"): # FT_GetPipeTimeout not in D3XX Linux library.
        Status, TimeOut = PyD3XX.FT_GetPipeTimeout(Device, Pipes[i])
        if (Status != PyD3XX.FT_OK) or (TimeOut != 100):
            print("FAILED TO SET AND/OR CONFIRM PIPE TIMEOUT TO 100: ABORTING")
            print("Status = " + PyD3XX.FT_STATUS_STR[Status] + " | Timeout = " +
str(TimeOut))
            exit()
    if(Status_SPT != PyD3XX.FT_OK):
        print("FAILED TO SET AND/OR CONFIRM PIPE TIMEOUT TO 100: ABORTING")
        print("Status = " + PyD3XX.FT_STATUS_STR[Status] + " | Timeout = " + str(TimeOut))
    else:
        print("Successfully set timeout of pipe " + str(i) + " to 100!")
```

State	Output
CHANNEL_COUNT = 1 and success.	Successfully set timeout of pipe 0 to 100! Successfully set timeout of pipe 1 to 100!

2.18 – FT_AbortPipe()

D3XX C API Equivalent = FT_AbortPipe()

Summary

Aborts the pending transfers for a pipe.

Parameters

Type	Name	Description
FT_Device	Device	The device whose FT_Pipe you want to abort the transfers of.
FT_Pipe	Pipe	The pipe you want to abort the transfer of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	LengthTransferred	The timeout value of the pipe (IN endpoint).

Example Code

Example Code

```
if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipeAsync(Device, 0, 1, Overlaps[1])
else:
    Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, Pipes[1], 1, Overlaps[1])
Status, BytesTransferred = PyD3XX.FT_GetOverlappedResult(Device, Overlaps[1], False)
if(Status == PyD3XX.FT_IO_INCOMPLETE):
    print("Waiting on input! Attempting FT_AbortPipe()...")
    Status = PyD3XX.FT_AbortPipe(Device, Pipes[1])
    if(Status != PyD3XX.FT_OK):
        print("FT_AbortPipe failed!")
    else:
        print("FT_AbortPipe succeeded!")
else:
    print(PyD3XX.FT_STATUS_STR[Status] + " | Encountered error or FT_ReadPipe was
successful.")
    ReadValue = format(int.from_bytes(ReadBuffer.Value(), "little"), "x").zfill(1*2)
    print("Channel 1 = 0x" + ReadValue)
```

State	Output
Waiting on input.	Waiting on input! Attempting FT_AbortPipe()... FT_AbortPipe succeeded!
Received 1 byte with value 0x01.	FT_OK | Encountered error or FT_ReadPipe was successful. Channel 1 = 0x01

2.19 – FT_GetDeviceDescriptor()

D3XX C API Equivalent = FT_GetDeviceDescriptor()

Summary

Get the USB device descriptor of a FT_Device object.

Parameters

Type	Name	Description
FT_Device	Device	The device whose USB device descriptor you want to retrieve.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_DeviceDescriptor	DeviceDescriptor	The USB device descriptor.

Example Code

Example Code

```
Status, DeviceDescriptor = PyD3XX.FT_GetDeviceDescriptor(Device)
#Status, DeviceDescriptor, BytesTransferred = PyD3XX.FT_GetDescriptor(Device,
PyD3XX.FT_DEVICE_DESCRIPTOR_TYPE, 0)
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET DEVICE DESCRIPTOR FOR DEVICE 0: ABORTING.")
    exit()
print("---| Device 0 Device Descriptor |---")
print("\tLength = " + str(DeviceDescriptor.bLength))
print("\tDescriptor Type = " + hex(DeviceDescriptor.bDescriptorType))
```

State: Success.

Output:

```
---| Device 0 Device Descriptor |---
        Length = 18
        Descriptor Type = 0x1
```

2.20 – FT_GetConfigurationDescriptor()

D3XX C API Equivalent = FT_GetConfigurationDescriptor()

Summary

Get the USB configuration descriptor of a FT_Device object.

Parameters

Type	Name	Description
FT_Device	Device	The device whose USB configuration descriptor you want to retrieve.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_ConfigurationDescriptor	ConfigurationDescriptor	The USB configuration descriptor.

Example Code

Example Code

```
Status, ConfigurationDescriptor = PyD3XX.FT_GetConfigurationDescriptor(Device)
#Status, ConfigurationDescriptor, BytesTransferred = PyD3XX.FT_GetDescriptor(Device,
PyD3XX.FT_CONFIGURATION_DESCRIPTOR_TYPE, 0)
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET THE CONFIGURATION DESCRIPTOR FOR DEVICE 0: ABORTING.")
    exit()
print("---| Device 0 Configuration Descriptor |---")
print("\tLength = " + str(ConfigurationDescriptor.bLength))
print("\tDescriptor Type = " + hex(ConfigurationDescriptor.bDescriptorType))
```

State: Success.

Output:

```
---| Device 0 Configuration Descriptor |---
        Length = 9
        Descriptor Type = 0x2
```

2.21 – FT_GetInterfaceDescriptor()

D3XX C API Equivalent = FT_GetInterfaceDescriptor()

Summary

Get the USB interface descriptor of a FT_Device object.

Parameters

Type	Name	Description
FT_Device	Device	The device whose USB interface descriptor you want to retrieve.
int	InterfaceIndex	The index of the interface descriptor you want to retrieve.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_InterfaceDescriptor	InterfaceDescriptor	The USB interface descriptor.

Example Code

Example Code

```
Status, Interface = PyD3XX.FT_GetInterfaceDescriptor(Device, 1) # Index 0 is for proprietary
protocols. Using 1 instead.
#Status, Interface, BytesTransferred = PyD3XX.FT_GetDescriptor(Device,
PyD3XX.FT_INTERFACE_DESCRIPTOR_TYPE, 1)
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET INTERFACE INFO OF [1]: ABORTING")
    exit()
print("---| Interface 1 |---")
print("\tLength = " + hex(Interface.bLength))
print("\tDescriptor Type = " + hex(Interface.bDescriptorType))
```

State: Success.

Output:

```
---| Interface 1 |---
        Length = 0x9
        Descriptor Type = 0x4
```

2.21B – FT_GetStringDescriptor()

D3XX C API Equivalent = FT_GetStringDescriptor()

Summary

Get the USB string descriptor of a FT_Device object.

Parameters

Type	Name	Description
FT_Device	Device	The device whose USB string descriptor you want to retrieve.
int	StringIndex	The index of the string descriptor you want to retrieve.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

FT_StringDescriptor	StringDescriptor	The USB string descriptor.

Example Code

Example Code

```
Status, StringDescriptor = PyD3XX.FT_GetStringDescriptor(Device, 2)
#Status, StringDescriptor, BytesTransferred = PyD3XX.FT_GetDescriptor(Device,
PyD3XX.FT_STRING_DESCRIPTOR_TYPE, 2)
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET THE STRING DESCRIPTOR FOR DEVICE 0: ABORTING.")
    exit()
print("---| Device 0 String[2] Descriptor |---")
print("\tLength = " + str(StringDescriptor.bLength))
print("\tDescriptor Type = " + hex(StringDescriptor.bDescriptorType))
print("\tString = '" + StringDescriptor.szString + "'")
```

State	Output
Success.	---| Device 0 String[2] Descriptor |--- Length = 56 Descriptor Type = 0x3 String = 'FTDI SuperSpeed-FIFO Bridge'

2.22 – FT_GetPipeInformation()

D3XX C API Equivalent = FT_GetPipeInformation()

Summary

Get an FT_Pipe object from the given open FT_Device object. The returned FT_Pipe object will only contain valid pipe information if the FT_Device object was previously opened by FT_Create() and is still open.

Interface 1 (data interface) is for the data read and write pipes. Pipes with an even index are write pipes and pipes with an odd index are read pipes.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to get an FT_Pipe from.
int	InterfaceIndex	The index of the interface you want to get a USB endpoint descriptor (FT_Pipe) from.
int	PipeIndex	The index of the pipe (FT_Pipe) you want to get from the given interface.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_Pipe	Pipe	An FT_Pipe object that can be used for read or write calls.

Example Code

Example Code

```
Pipes = {}
for i in range(CHANNEL_COUNT * 2): # Print info for all pipes for interface 1.
    Status, Pipes[i] = PyD3XX.FT_GetPipeInformation(Device, 1, i) # Get pipe information.
    if Status != PyD3XX.FT_OK:
        print("FAILED TO GET PIPE INFO OF [1," + str(i) +"]: ABORTING")
        exit()
    print("---| Pipe " + str(i) + " |---")
    print("\tPipe Type = " + hex(Pipes[i].PipeType))
    print("\tPipe ID = " + hex(Pipes[i].PipeID))
    print("\tPipe MaxPacketSize = " + str(Pipes[i].MaximumPacketSize))
    print("\tPipe Interval = " + str(Pipes[i].Interval))
```

State

```
CHANNEL_COUNT = 1
```

Output

```
---| Pipe 0 |---
        Pipe Type = 0x2
        Pipe ID = 0x2
        Pipe MaxPacketSize = 0
        Pipe Interval = 4
---| Pipe 1 |---
        Pipe Type = 0x2
        Pipe ID = 0x82
        Pipe MaxPacketSize = 0
        Pipe Interval = 4
```

2.23 – FT_GetDescriptor()

D3XX C API Equivalent = FT_GetDescriptor()

Summary

Get a USB descriptor from a FT_Device object. Returns any one of the following descriptor objects: FT_DeviceDescriptor, FT_ConfigurationDescriptor, FT_StringDescriptor, FT_InterfaceDescriptor.

WARNING: this is not a full equivalent of FT_GetDescriptor() as it only returns four common descriptors. This function's implementation will likely change in the future to closer match the C library equivalent's ability to get uncommon USB descriptors. Use the dedicated functions FT_GetDeviceDescriptor, FT_GetConfigurationDescriptor, FT_GetInterfaceDescriptor, FT_GetStringDescriptor, and FT_GetPipeInformation instead.

Parameters

Type	Name	Description
FT_Device	Device	The device whose USB descriptor you want to retrieve.
int	DescriptorType	The type of descriptor you want to retrieve.
int	Index	The index of the descriptor you want to retrieve.

Return Values

Type	Name	Description
int	Status	FT_OK if successful. Status is always returned.

Condition: DescriptorType = FT_DEVICE_DESCRIPTOR_TYPE		
FT_DeviceDescriptor	DeviceDescriptor	The USB device descriptor.
Condition: DescriptorType = FT_CONFIGURATION_DESCRIPTOR_TYPE		
FT_ConfigurationDescriptor	ConfigurationDescriptor	The USB configuration descriptor.
Condition: DescriptorType = FT_STRING_DESCRIPTOR_TYPE		
FT_StringDescriptor	StringDescriptor	The USB string descriptor.
Condition: DescriptorType = FT_INTERFACE_DESCRIPTOR_TYPE		
FT_InterfaceDescriptor	InterfaceDescriptor	The USB interface descriptor.
Condition: Always		
int	LengthTransferred	Number of bytes transferred to the descriptor buffer.

Example Code

Example Code

```
#Status, StringDescriptor = PyD3XX.FT_GetStringDescriptor(Device, 2)
Status, StringDescriptor, BytesTransferred = PyD3XX.FT_GetDescriptor(Device,
PyD3XX.FT_STRING_DESCRIPTOR_TYPE, 2)
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET THE STRING DESCRIPTOR FOR DEVICE 0: ABORTING.")
    exit()
print("---| Device 0 String[2] Descriptor |---")
print("\tLength = " + str(StringDescriptor.bLength))
print("\tDescriptor Type = " + hex(StringDescriptor.bDescriptorType))
print("\tString = '" + StringDescriptor.szString + "'")
```

State: Success.

Output:

```
---| Device 0 String[2] Descriptor |---
        Length = 56
        Descriptor Type = 0x3
        String = 'FTDI SuperSpeed-FIFO Bridge'
```

2.24 – FT_ControlTransfer()

D3XX C API Equivalent = FT_ControlTransfer()

Summary

Transmits control data over the default control endpoint.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to do a control transfer with.
FT_SetupPacket	SetupPacket	The 8-byte setup packet.
FT_Buffer	Buffer	The buffer used for data transfer.
int	BufferLength	The length of the buffer used for data transfer.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	LengthTransferred	The number of bytes transferred.

Example Code

Example Code

```
Buffer = PyD3XX.FT_Buffer(1)
SetupPacket = PyD3XX.FT_SetupPacket
SetupPacket.RequestType = int("10000001", 2)
SetupPacket.Request = 10
SetupPacket.Value = 0
SetupPacket.Index = 1
SetupPacket.Length = 1
Status, DataWritten = PyD3XX.FT_ControlTransfer(Device, SetupPacket, Buffer, 1)
if Status != PyD3XX.FT_OK:
    print("WARNING: FAILED TO GET ALTERNATE SETTINGS OF INTERFACE 1 FROM DEVICE 0.")
print("Alternate Settings of Interface 1: " + hex(int.from_bytes(Buffer.Value(), "little")))
```

State	Output
Success.	Alternate Settings of Interface 1: 0x0

2.25 – FT_GetVIDPID()

D3XX C API Equivalent = FT_GetVIDPID()

Summary

Get the vendor ID (VID) and product ID (PID) of an FT_Device object. The device must be opened beforehand.

Parameters

Type	Name	Description
FT_Device	Device	The device you want get the VID & PID of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	VID	The vendor ID of the given device.
int	PID	The product ID of the given device.

Example Code

Example Code

```
Status, Device = PyD3XX.FT_GetDeviceInfoDetail(0) # Get info of a device at index 0.
Status = PyD3XX.FT_Create(0, PyD3XX.FT_OPEN_BY_INDEX, Device) # Open the device we're using.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO OPEN DEVICE: ABORTING")
    exit()
Status, VID, PID = PyD3XX.FT_GetVIDPID(Device)
print("Device 0 (VID:PID) = " + format(VID, "x").zfill(4) + ":" + format(PID, "x").zfill(4))
```

State	Output
Success.	Device 0 (VID:PID) = 0403:601e

2.26 – FT_EnableGPIO()

D3XX C API Equivalent = FT_EnableGPIO()

Summary

Enable GPIO mode and set the input/output direction of the two GPIO pins.

Parameters

Type	Name	Description
FT_Device	Device	The device you want enable the GPIO of.
int	EnableMask	Select which GPIO pins to enable. Bits 31-2 are ignored. 0 = ignore. 1 = enable.
int	DirectionMask	Select which input/output direction the GPIO pins should have. Bits 31-2 are ignored. 0 = input. 1 = output.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_EnableGPIO(Device, int("11", 2), int("11", 2))
if Status != PyD3XX.FT_OK:
    print("WARNING: FAILED TO ENABLE GPIO OF DEVICE 0")
else:
    print("Successfully enabled GPIO[1:0] as outputs.");
```

State	Output
Success.	Successfully enabled GPIO[1:0] as outputs.

2.27 – FT_WriteGPIO()

D3XX C API Equivalent = FT_WriteGPIO()

Summary

Set the output state of the GPIO0 and GPIO1 pins. Make sure FT_EnableGPIO() is used beforehand to make the GPIO pins outputs.

Parameters

Type	Name	Description
FT_Device	Device	The device whose GPIO pins you want to write to.
int	SelectMask	Select which GPIO pins to change the output state of. Bit 0 = GPIO0. Bit 1 = GPIO1. 1 = change output state. 0 = ignore.
int	Data	Data to write to the GPIO pins. Bit 0 = GPIO0. Bit 1 = GPIO1.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_WriteGPIO(Device, int("11", 2), int("11", 2))
if Status != PyD3XX.FT_OK:
    print("WARNING: FAILED TO WRITE GPIO OF DEVICE 0")
else:
    print("Successfully wrote 0b11 to GPIO pins!");
```

State	Output
Success.	Successfully wrote 0b11 to GPIO pins!

2.28 – FT_ReadGPIO()

D3XX C API Equivalent = FT_ReadGPIO()

Summary

Read the state of the GPIO0 and GPIO1 pins. Make sure FT_EnableGPIO() is used beforehand to make the GPIO pins inputs.

Parameters

Type	Name	Description
FT_Device	Device	The device whose GPIO pins you want to read from.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	Data	Data read from the GPIO pins. Bit 0 = GPIO0. Bit 1 = GPIO1.

Example Code

Example Code

```
Status, GPIO_Data = PyD3XX.FT_ReadGPIO(Device)
if Status != PyD3XX.FT_OK:
    print("WARNING: FAILED TO READ GPIO OF DEVICE 0")
else:
    print("GPIO DATA = 0b" + format(GPIO_Data, "02b"))
```

State	Output
GPIO[1:0] = 0b10	GPIO DATA = 0b10

2.29 – FT_SetGPIOPull()

D3XX C API Equivalent = FT_SetGPIOPull()

Summary

Set the internal resistor pull state of the GPIO0 and GPIO1 pins high, low, or hi-Z. This only works for revision B FT60X devices or newer.

Parameters

Type	Name	Description
FT_Device	Device	The device whose GPIO pins you want to change the pull state of.
Int	SelectMask	Select which GPIO pins to change the pull state of. Bit 0 = GPIO0.

		Bit 1 = GPIO1. 1 = change pull state. 0 = ignore.
int	PullMask	Control what pull state the GPIO pins are set to. PullMask[1:0] = GPIO0 pull state, PullMask[3:2] = GPIO1 pull state. 0b00 = 50k ohm pull-down (default) 0b01 = Hi-Z 0b10 = 50k ohm pull-up 0b11 = Hi-Z

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_SetGPIOPull(Device, int("11", 2), int("1111", 2))
if Status != PyD3XX.FT_OK:
    print("WARNING: FAILED TO SET GPIO PULL OF DEVICE 0")
else:
    print("Successfully set GPIO pull to hi-Z!");
```

State	Output
Success.	Successfully set GPIO pull to hi-Z!

2.30 – FT_SetNotificationCallback()

D3XX C API Equivalent = FT_SetNotificationCallback()

Summary

Set a callback function that will be called when any IN endpoint with no current ongoing read requests receive data. If notification messages are enabled for a specific channel, never make a FT_ReadPipe call unless it is within the callback function for the exact number of bytes available in the IN endpoint.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to set the callback notification function for.
typing.Callable[[int, int, int], None]	CallbackFunction	Your callback function.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status, ReadPipeCH1 = PyD3XX.FT_GetPipeInformation(Device, 1, 1)
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET READ PIPE INFO OF CH1: ABORTING")
    exit()
def CallBackFunction(CallbackType: int, PipeID_GPIO0: int | bool, Length_GPIO1: int | bool):
    global NotificationCount
    NotificationCount += 1
    print("Received " + str(NotificationCount) + " notifications!")
    if(CallbackType == PyD3XX.E_FT_NOTIFICATION_CALLBACK_TYPE_DATA):
        print("CBF: You have " + str(Length_GPIO1) + " bytes to read at pipe " +
hex(PipeID_GPIO0) + "!")
        if(NotificationCount != 5): # Disable callback function.
            if(PyD3XX.Platform == "linux") or (PyD3XX.Platform == "darwin"):
                Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, int("0x82", 16),
Length_GPIO1, 0)
            else:
                Status, ReadBuffer, BytesRead = PyD3XX.FT_ReadPipe(Device, ReadPipeCH1,
Length_GPIO1, PyD3XX.NULL)
        else:
            Status = PyD3XX.FT_ClearNotificationCallback(Device) # Clear callback function so
we don't get called again.
            if Status != PyD3XX.FT_OK:
                print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO CLEAR CALLBACK
FUNCTION OF DEVICE 0")
    elif(CallbackType == PyD3XX.E_FT_NOTIFICATION_CALLBACK_TYPE_GPIO):
        print("CBF: GPIO 0 Status = " + str(PipeID_GPIO0) + " | GPIO 1 Status = " +
str(Length_GPIO1))
    return None

PyD3XX.FT_SetNotificationCallback(Device, CallBackFunction)
if Status != PyD3XX.FT_OK:
    print("FAILED TO SET CALLBACK FUNCTION OF DEVICE 0: ABORTING")
    exit()
print("This program will exit after at least 5 notifications or 5 seconds.")
Loops = 0
while((NotificationCount != 5) and not(Loops == 5)):
    time.sleep(1)
    Loops += 1;
print("Total Notifications Received: " + str(NotificationCount))
print("Seconds Passed: " + str(Loops))
```

State	Output
Channel 1 received five master writes while no read requests were ongoing.	This program will exit after at least 5 notifications or 5 seconds. Received 1 notifications! CBF: You have 4 bytes to read at pipe 0x82! Received 2 notifications! CBF: You have 2 bytes to read at pipe 0x82! Received 3 notifications! CBF: You have 4 bytes to read at pipe 0x82! Received 4 notifications! CBF: You have 4 bytes to read at pipe 0x82! Received 5 notifications! CBF: You have 4 bytes to read at pipe 0x82! Total Notifications Received: 5 Seconds Passed: 1

2.31 – FT_ClearNotificationCallback()

D3XX C API Equivalent = FT_ClearNotificationCallback()

Summary

Clears the notification callback function set by FT_SetNotificationCallback.

Parameters

Type	Name	Description
FT_Device	Device	The device you want clear the callback notification function for.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code:

```
Status = PyD3XX.FT_ClearNotificationCallback(Device) # Clear callback function.
    print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO CLEAR CALLBACK FUNCTION OF DEVICE 0")
```

State	Output
Success.	
Failure.	FT_INVALID_HANDLE | WARNING: FAILED TO CLEAR CALLBACK FUNCTION OF DEVICE 0

2.32 – FT_GetChipConfiguration()

D3XX C API Equivalent = FT_GetChipConfiguration()

Summary

Get the chip configuration of the given FT_Device object.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to get the chip configuration of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
FT_60XCONFIGURATION	Configuration	The chip configuration of the given device.

Example Code

Example Code

```
Status, ChipConfiguration = PyD3XX.FT_GetChipConfiguration(Device) # Get configuration.
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET CHIP CONFIGURATION OF Device 0: ABORTING.")
    exit()
print("Device 0 Description: '" + ChipConfiguration.StringDescriptors[1] + "'")
```

State	Output
Success.	Device 0 Description: 'TEST 1'

2.33 – FT_SetChipConfiguration()

D3XX C API Equivalent = FT_SetChipConfiguration()

Summary

Set the chip configuration of the given FT_Device object.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to set the chip configuration of.
FT_60XCONFIGURATION	Configuration	The chip configuration you want to set.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status, ChipConfiguration = PyD3XX.FT_GetChipConfiguration(Device) # Get chip configuration.
if Status != PyD3XX.FT_OK:
    print("FAILED TO GET CHIP CONFIGURATION OF Device 0: ABORTING.")
    exit()
print("Device 0 Description: '" + ChipConfiguration.StringDescriptors[1] + "'")
if(ChipConfiguration.StringDescriptors[1] == "TEST 0"):
    ChipConfiguration.StringDescriptors[1] = "TEST 1"
else:
    ChipConfiguration.StringDescriptors[1] = "TEST 0"
Status = PyD3XX.FT_SetChipConfiguration(Device, ChipConfiguration)
if Status != PyD3XX.FT_OK:
    print("WARNING: FAILED TO SET CHIP CONFIGURATION OF DEVICE 0")
```

State	Output
Success.	Device 0 Description: 'TEST 1'
Success again.	Device 0 Description: 'TEST 0'

2.34 – FT_IsDevicePath()

D3XX C API Equivalent = FT_IsDevicePath()
WARNING – This function is exclusive to Windows systems.

Summary

Returns FT_OK if the given device path matches the device path of the device handle.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to check the path of.
str	DevicePath	The device path you want to compare.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_IsDevicePath(Device, DevicePath)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CONFIRM DEVICE PATH.")
else:
    print("SUCCESSFULLY CONFIRMED DEVICE PATH.")
```

State	Output
Success.	SUCCESSFULLY CONFIRMED DEVICE PATH.
Incorrect path.	FT_INCORRECT_DEVICE_PATH | FAILED TO CONFIRM DEVICE PATH.

2.35 – FT_GetDriverVersion()

D3XX C API Equivalent = FT_GetDriverVersion()

Summary

Returns the D3XX kernel driver version number in 4 bytes formatted in 4-bit Binary-Coded Decimal (BCD). Example: 0x01457000 = 1.45.70.0.

Parameters

Type	Name	Description
FT_Device	Device	The device you want get the kernel driver version of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	DriverVersion	The driver version encoded in 4-bit BCD format.

Example Code

Example Code:

```
Status, DriverVersion = PyD3XX.FT_GetDriverVersion(Device)
print("Device 0 Driver Version: " + format((DriverVersion & int("0xFF000000", 16)) >> 24,
'x') + '.' \
    + format((DriverVersion & int("0x00FF0000", 16)) >> 16, 'x') + '.' \
    + format((DriverVersion & int("0x0000FF00", 16)) >> 8, 'x') + '.' \
    + format(DriverVersion & int("0x000000FF", 16), 'x')
    )
```

State: Version = 0x01030010

Output:

```
Device 0 Driver Version: 1.3.0.10
```

2.35B – GetDriverVersion()

D3XX C API Equivalent = NONE. PyD3XX exclusive.

Summary

Returns the D3XX kernel driver version number as a string. Example: 0x01457000 = "1.45.70.0".

This is meant as an easier to use form of FT_GetDriverVersion().

Parameters

Type	Name	Description
FT_Device	Device	The device you want get the kernel driver version of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
str	DriverVersion	The driver version as a string.

Example Code

Example Code	
	`Status, DriverVersion = PyD3XX.GetDriverVersion(Device)` `print("Device 0 Driver Version: " + DriverVersion)`

State	Output
Version = 0x01030010	Device 0 Driver Version: 1.3.0.10

2.36 – FT_GetLibraryVersion()

D3XX C API Equivalent = FT_GetLibraryVersion()

Summary

Returns the D3XX user driver library version number in 4 bytes formatted in 4-bit Binary-Coded Decimal (BCD) for Windows. Example: 0x01457000 = 1.45.70.0.

For Linux/macOS the library version is the version of libusb installed on the system.

Parameters

None

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	DriverVersion	The driver library version encoded in 4-bit BCD format.

Example Code

```
Status, LibraryVersion = PyD3XX.FT_GetLibraryVersion()
print("Device 0 Library Version: " + format((LibraryVersion & int("0xFF000000", 16)) >> 24,
'x') + '.' \
    + format((LibraryVersion & int("0x00FF0000", 16)) >> 16, 'x') + '.' \
    + format((LibraryVersion & int("0x0000FF00", 16)) >> 8, 'x') + '.' \
    + format(LibraryVersion & int("0x000000FF", 16), 'x')
    )
```

State	Output
Version = 0x01030008	Device 0 Library Version: 1.3.0.8

2.36B – GetLibraryVersion()

D3XX C API Equivalent = NONE. PyD3XX exclusive.

Summary

Returns the D3XX user driver library version number as a string. Example: 0x01457000 = "1.45.70.0".

This is meant as an easier to use form of FT_GetLibraryVersion().

Parameters
None

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
str	DriverVersion	The driver library version as a string.

Example Code

Example Code:
```
Status, LibraryVersion = PyD3XX.GetLibraryVersion()
print("Device 0 Library Version: " + LibraryVersion)
```

State	Output
Version = 0x01030008	Device 0 Library Version: 1.3.0.8

2.37 – FT_CycleDevicePort()
D3XX C API Equivalent = FT_CycleDevicePort()

Summary
Makes the host system power cycle the device port so that the device re-enumerates.

Parameters

Type	Name	Description
FT_Device	Device	The device whose port you want to power cycle.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code:
```
Status = PyD3XX.FT_CycleDevicePort(Device)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO CYCLE PORT OF DEVICE 0")
else:
    print("Successfully cycled port of Device 0!")
```

State	Output
Success.	Successfully cycled port of Device 0!

2.37B – FT_ResetDevicePort()
D3XX C API Equivalent = FT_ResetDevicePort()

Summary

Makes the host system reset the device port so that the device re-enumerates.

Parameters

Type	Name	Description
FT_Device	Device	The device whose port you want to reset.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status = PyD3XX.FT_ResetDevicePort(Device)
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | WARNING: FAILED TO RESET PORT OF DEVICE 0")
else:
    print("Successfully reset port of Device 0!")
```

State	Output
Success.	Successfully reset port of Device 0!

2.38 – FT_SetSuspendTimeout()

D3XX C API Equivalent = FT_SetSuspendTimeout()
WARNING – This function is exclusive to Windows systems.

Summary

Sets the USB Selective suspend timeout for the given FT_Device. By default with unmodified drivers, devices have the suspend feature enabled with an idle timeout of 10 seconds. If the FT_Device is closed and re-opened, then the idle timeout will reset to the driver default.

If you want your device to always output a CLK signal, you should disable USB selective suspend. FT_SetSuspendTimeout is only available on Windows systems. This function will fail if the notification feature is enabled.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to set the suspend timeout value of.
int	Timeout	The new USB selective suspend idle timeout value in seconds. If set to 0, then USB selective suspend will be disabled.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.

Example Code

Example Code

```
Status, ChipConfiguration = PyD3XX.FT_GetChipConfiguration(Device) # Get chip configuration.
CallbackNotificationsEnabled = (ChipConfiguration.OptionalFeatureSupport &
PyD3XX.CONFIGURATION_OPTIONAL_FEATURE_ENABLENOTIFICATIONMESSAGE_INCHALL) != 0
if(PyD3XX.Platform != "windows"):
    print("FT_SetSuspendTimeout is only available on Windows!")
    exit()
if(CallbackNotificationsEnabled != False): # Can only disable suspend timeout if callback
notifications are disabled.
    print("WARNING: Callback notifications are enabled!")
Status = PyD3XX.FT_SetSuspendTimeout(Device, 0) # Disable suspend timeout.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO DISABLE SUSPEND MODE: ABORTING")
    exit()
Status, SuspendTimeout = PyD3XX.FT_GetSuspendTimeout(Device)
if (Status != PyD3XX.FT_OK) or (SuspendTimeout != 0):
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CONFIRM SUSPEND MODE: ABORTING")
    exit()
print("Successfully disabled suspend mode!");
```

State	Output
Success.	Successfully disabled suspend mode!
Not on Windows.	FT_SetSuspendTimeout is only available on Windows!
Notifications are enabled.	WARNING: Callback notifications are enabled! FT_OTHER_ERROR | FAILED TO DISABLE SUSPEND MODE: ABORTING

2.39 – FT_GetSuspendTimeout()

D3XX C API Equivalent = FT_GetSuspendTimeout()
WARNING – This function is exclusive to Windows systems.

Summary

Returns the idle timeout value for USB Selective suspend for the given FT_Device object.

FT_GetSuspendTimeout is only available on Windows systems. This function will fail if the notification feature is enabled.

Parameters

Type	Name	Description
FT_Device	Device	The device you want to get the suspend timeout value of.

Return Values

Type	Name	Description
int	Status	FT_OK if successful.
int	Timeout	The USB selective suspend idle timeout value in seconds. If 0, then USB selective suspend is disabled.

Example Code

Example Code

```
Status = PyD3XX.FT_SetSuspendTimeout(Device, 0) # Disable suspend timeout.
if Status != PyD3XX.FT_OK:
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO DISABLE SUSPEND MODE: ABORTING")
    exit()
Status, SuspendTimeout = PyD3XX.FT_GetSuspendTimeout(Device)
if (Status != PyD3XX.FT_OK) or (SuspendTimeout != 0):
    print(PyD3XX.FT_STATUS_STR[Status] + " | FAILED TO CONFIRM SUSPEND MODE: ABORTING")
    exit()
print("Successfully disabled suspend mode!");
```

State	Output
Success.	Successfully disabled suspend mode!

3 – PyD3XX Classes

All the classes defined in PyD3XX.

3.1 – FT_Buffer

D3XX C API Equivalent = NONE. PyD3XX exclusive.

Summary

A buffer that holds data to be written or data that was received for write/read functions like FT_WritePipe() and FT_ReadPipe().

Attributes

None

Methods

Name	Parameters	Description
__init__	Size: int	Construct an FT_Buffer that can hold Size bytes.
from_int	Integer: int	Construct an FT_Buffer from an integer.
from_str	String: str	Construct an FT_Buffer from an ASCII string.
from_bytearray	ByteArray: bytearray	Construct an FT_Buffer from a bytearray.
from_bytes	Bytes: bytes	Construct an FT_Buffer from bytes.
Value	None	Return the data of the buffer in a bytearray.

3.2 – FT_Device

D3XX C API Equivalent = FT_DEVICE_LIST_INFO_NODE / FT_HANDLE

Summary

A class that holds the same information about a device as FT_DEVICE_LIST_INFO_NODE but is also used as a replacement for FT_HANDLE for function calls. You must initialize/obtain an FT_Device from functions like FT_GetDeviceInfoDetail or FT_GetDeviceInfoList for an FT_Device object to be valid and openable.

Attributes

Type	Name	Description		
int | str	Handle	The value of a valid FT_HANDLE as an integer or a string of an FT_ERROR like "FT_OTHER_ERROR".		
int	Flags	A bitmask showing the status of the device.		
		0b001	FT_FLAGS_OPENED	Device is opened by another program.
		0b010	FT_FLAGS_HISPEED	Device is operating at USB Hi-Speed. Note: Device could also be operating at USB Full Speed. If MaximumPacketSize from FT_Pipe is 512, it's USB Hi-Speed. If it's 64, it's USB Full Speed.
		0b100	FT_FLAGS_SUPERSPEED	Device is operating at USB SuperSpeed.
int	Type	Represents what type of FT60X device the FT_Device is.		
		0d3	FT_DEVICE_UNKNOWN	Device type is unknown.
		0d600	FT_DEVICE_600	Device is an FT600.
		0d601	FT_DEVICE_601	Device is an FT601.
int	LocID	The device's location ID.		
str	SerialNumber	The device's serial number as a string.		
str	Description	The device's description as a string.		

Methods
None

3.3 – FT_DeviceDescriptor
D3XX C API Equivalent = FT_DEVICE_DESCRIPTOR

Summary
The USB device descriptor retrieved from an FT_Device.

Attributes
Type	Name	Description
int	BLength	The length of the device descriptor itself.
int	bDescriptorType	The type of the descriptor itself.
int	bcdUSB	The version of the USB specification the device conforms to. 0x0200 = USB 2.0, 0x0310 = USB 3.1, etc.
int	bDeviceClass	The device's USB base class code. More Info: https://www.usb.org/defined-class-codes
int	bDeviceSubClass	The device's USB sub class code. More Info: https://www.usb.org/defined-class-codes
int	bDeviceProtocol	The device's USB device protocol. More Info: https://www.usb.org/defined-class-codes
int	bMaxPacketSize0	The max packet size of endpoint zero.
int	idVendor	The device's vendor ID (VID).
int	idProduct	The device's product ID (PID).
int	bcdDevice	The device's device-defined revision number.
int	iManufacturer	The device's device-defined index of the string descriptor containing the name of the device's manufacturer.
int	iProduct	The device's device-defined index of the string descriptor containing the device's description.
int	iSerialNumber	The device's device-defined index of the string descriptor containing the device's serial number.

int	bNumConfigurations	The device's total number of possible configurations.

Methods
None

3.4 – FT_ConfigurationDescriptor
D3XX C API Equivalent = FT_CONFIGURATION_DESCRIPTOR

Summary
The USB configuration descriptor retrieved from an FT_Device. FTDI D3XX devices support only one USB configuration.

Attributes

Type	Name	Description
int	BLength	The length of the configuration descriptor itself.
int	bDescriptorType	The type of the descriptor itself.
int	wTotalLength	The number of bytes needed to contain the full configuration descriptor. This length accounts for all interface, endpoint, class, and/or vendor-specific descriptors that are returned with the configuration descriptor.
int	bNumInterfaces	The total number of interfaces supported by the configuration.
int	bConfigurationValue	The value used to select a configuration.
int	iConfiguration	The device's device-defined index of the string descriptor for this configuration.
int	bmAttributes	A bitmap describing the behavior of the configuration.
	bmAttributes[4:0]	Bits 4-0. Reserved.
	bmAttributes[5]	0b1 = The configuration supports remote wakeup.
	bmAttributes[6]	0b1 = The configuration is self-powered.
	bmAttributes[7]	0b1 = The configuration is bus-powered.
int	MaxPower	Power requirements of the device represented in 2mA units typically. Example: MaxPower = 0d50 means device has a power requirement of 100mA (50 * 2). Warning: For D3XX, MaxPower appears to represent power in units of 8mA and not 2mA. MaxPower = 0d62 means device has a power requirement of 496mA (62 * 8).

Methods
None

3.5 – FT_InterfaceDescriptor
D3XX C API Equivalent = FT_INTERFACE_DESCRIPTOR

Summary
The USB interface descriptor retrieved from an FT_Device.

Attributes

Type	Name	Description
int	BLength	The length of the interface descriptor itself.
int	bDescriptorType	The type of the descriptor itself.

int	bInterfaceNumber	The index number of the interface.
int	bAlternateSetting	The index number of the alternate setting of the interface.
int	bNumEndpoints	The number of endpoints used by the interface, excluding the status endpoint.
int	bInterfaceClass	The class code of the device.
int	bInterfaceSubClass	The sub class code of the device.
int	bInterfaceProtocol	The protocol code of the device.
int	iInterface	The device's device-defined index of the string descriptor containing the device's description.

Methods
None

3.6 – FT_StringDescriptor
D3XX C API Equivalent = FT_STRING_DESCRIPTOR

Summary
The USB string descriptor retrieved from an FT_Device. Can hold a device, configuration, interface, class, vendor, or endpoint string descriptor.

Attributes
Type	Name	Description
int	BLength	The length of the string descriptor itself.
int	bDescriptorType	The type of the descriptor itself.
str	szString	The string for the requested descriptor.

Methods
None

3.7 – FT_EndpointDescriptor
D3XX C API Equivalent = FT_ENDPOINT_DESCRIPTOR

Summary
The USB endpoint descriptor retrieved from an FT_Device.

Attributes
Type	Name	Description	
int	bLength	The length of the endpoint descriptor itself.	
int	bDescriptorType	The type of the descriptor itself.	
int	bEndpointAddress	The USB-defined endpoint address.	
		bEndpointAddress[3:0]	The endpoint number.
		bEndpointAddress[6:4]	Reserved
		bEndpointAddress[7]	0 = OUT endpoint 1 = IN endpoint
int	bmAttributes	The attributes of the endpoint.	
		bmAttributes[1:0]	0b00 = This is a Control endpoint. 0b01 = This is an Isochronous endpoint. 0b10 = This is a Bulk endpoint. 0b11 = This is an Interrupt endpoint.

		bmAttributes[3:2]	0b00 = No synchronization. Bits 5-4 are zero. 0b01 = Asynchronous. 0b10 = Adaptive. 0b11 = Synchronous.
		bmAttributes[5:4]	0b00 = Used for data. 0b01 = Used for feedback. 0b10 = Used for implicit feedback. 0b11 = Reserved.
		bmAttributes[7:6]	Reserved.
int	wMaxPacketSize	The maximum packet size for this endpoint.	
int	bInterval	The polling interrupt for isochronous and interrupt endpoints.	

Methods
None

3.8 – FT_Pipe
D3XX C API Equivalent = FT_PIPE_INFORMATION / ucPipeID

Summary
An FT_Device's IN or OUT endpoint. Used as an equivalent for ucPipeID in read/write functions like FT_ReadPipe() and FT_WritePipe(). FT_Pipe is the read or write endpoint for one of the FT60X's channels.

Attributes

Type	Name	Description		
int	PipeType	The type of the pipe.		
		0d0	FTPipeTypeControl	This is a Control pipe.
		0d1	FTPipeTypeIsochronous	This is an Isochronous pipe.
		0d2	FTPipeTypeBulk	This is a Bulk pipe.
		0d3	FTPipeTypeInterrupt	This is an Interrupt pipe.
int	PipeId	The pipe's ID.		
		PipeId[3:0]	The pipe's number.	
		PipeId[6:4]	Reserved	
		PipeId [7]	0 = This is a write (OUT) pipe. 1 = This is a read (IN) pipe.	
int	MaximumPacketSize	The maximum packet size for this pipe.		
int	Interval	The pipe's interrupt interval if used for callbacks.		

Methods
None

3.9 – FT_Overlapped
D3XX C API Equivalent = OVERLAPPED

Summary
The equivalent of the OVERLAPPED structure used for asynchronous I/O operations.

Attributes

Type	Name	Description

int	Internal	The status code for the I/O request.
int	InternalHigh	The number of bytes transferred for a successful I/O request.
int	DummyUnion	The address of the dummy union.
int	DummyUnion_Offset	The low-order part of the file position specified by the user to start the I/O request.
int	DummyUnion_OffsetHigh	The high-order part of the file position specified by the user to start the I/O request.
int	DummyUnion_Pointer	Reserved. Do not use.
int	Handle	A handle to an event signaled by the system when the I/O operation has completed.

Methods
None

3.10 – FT_SetupPacket
D3XX C API Equivalent = FT_SETUP_PACKET

Summary
The equivalent of a USB setup packet to perform standard device requests. You can find more information about setup packets and standard device requests in official USB specifications.

Attributes

Type	Name	Description
int	RequestType	The type of request to be performed.
int	Request	The request code.
int	Value	A request-specific value.
int	Index	A request-specific value.
int	Length	The number of bytes to transfer.

Methods
None

3.11 – FT_60XCONFIGURATION
D3XX C API Equivalent = FT_60XCONFIGURATION

Summary
Holds the chip configuration read from or to be to written to a FT60X device.

Attributes

Type	Name	Description	
int	VendorID	The vendor ID. VID	
int	ProductID	The product ID. PID	
list[str]	StringDescriptors	A list of string descriptors which are encoded in ASCII.	
		StringDescriptors[0]	The manufacturer string, can hold a max of 15 characters.
		StringDescriptors[1]	The description string, can hold a max of 31 characters.
		StringDescriptors[2]	The serial number string, can hold a max of 15 characters.

int	bInterval	The minimum latency in 2^(bInterval-1) USB frames. This is the minimum latency for notification callbacks and GPIO reads. The minimum value is 1, and the maximum value is 16. Example: bInterval=15 means a latency of 2^14 USB frames. Since each frame is 125us, that is a minimum latency of 2.048 seconds (2^14 * 125us).
int	PowerAttributes	A bitmap describing the behavior of the configuration. PowerAttributes[4:0] — Bits 4-0. Reserved. PowerAttributes[5] — 0b1 = The device supports remote wakeup. PowerAttributes[6] — 0b1 = The device is self-powered. PowerAttributes[7] — 0b1 = The device is bus-powered.
int	PowerConsumption	Power requirements of the device represented in 8mA units. PowerConsumption = 0d62 means device has a max power requirement of 496mA (62 * 8).
int	Reserved2	Reserved.
int	FIFOClock	The clock speed of the FIFO. 0d0 = 100 MHz. 0d1 = 66 MHz.
int	FIFOMode	The mode of the FIFO. 0d0 = 245 Mode. 0d1 = 600 Mode.
int	ChannelConfig	The channel configuration. 0x0 = 4 Channels. 0x1 = 2 Channels. 0x2 = 1 Channel. 0x3 = 1 OUT Pipe. 0x4 = 1 IN Pipe.
int	OptionalFeatureSupport	A bitmask representing support for optional features.
int	BatteryChargingGPIOConfig	A bitmask to disable or set GPIO states that indicate what power source type was detected. 0d0 = Disabled. SDP = Standard Downstream Port. CDP = Charging Downstream Port. DCP = Dedicated Charging Port. BatteryChargingGPIOConfig[1:0] — GPIO default state. BatteryChargingGPIOConfig[3:2] — SDP detected state. BatteryChargingGPIOConfig[5:4] — CDP detected state. BatteryChargingGPIOConfig[7:6] — DCP detected state.
int	FlashEEPROMDetection	Read only bit mask.
int	MSIO_Control	Controls the drive strength of the FIFO pins. Default value is 0x010800. Make sure to only alter the first byte of data. Byte0[3:0] — Data pins' drive strength. 0d0 = 50 Ohm 0d1 = 35 Ohm 0d2 = 25 Ohm 0d3 = 18 Ohm Byte0[7:4] — CLK pin's drive strength. 0d0 = 50 Ohm 0d1 = 35 Ohm 0d2 = 25 Ohm

<table>
<tr><td></td><td></td><td></td><td colspan="2">0d3 = 18 Ohm</td></tr>
<tr><td>int</td><td>GPIO_Control</td><td colspan="3">Controls the drive strength of the GPIO pins.
<table><tr><th>GPIO_Control</th><th>GPIO0 Drive Strength</th><th>GPIO1 Drive Strength</th></tr><tr><td>0x0000</td><td>50 Ohm</td><td>50 Ohm</td></tr><tr><td>0x0100</td><td>35 Ohm</td><td>50 Ohm</td></tr><tr><td>0x0200</td><td>25 Ohm</td><td>50 Ohm</td></tr><tr><td>0x0300</td><td>18 Ohm</td><td>50 Ohm</td></tr><tr><td>0x0400</td><td>50 Ohm</td><td>35 Ohm</td></tr><tr><td>0x0500</td><td>35 Ohm</td><td>35 Ohm</td></tr><tr><td>0x0600</td><td>25 Ohm</td><td>35 Ohm</td></tr><tr><td>0x0700</td><td>18 Ohm</td><td>35 Ohm</td></tr><tr><td>0x0800</td><td>50 Ohm</td><td>25 Ohm</td></tr><tr><td>0x0900</td><td>35 Ohm</td><td>25 Ohm</td></tr><tr><td>0x0A00</td><td>25 Ohm</td><td>25 Ohm</td></tr><tr><td>0x0B00</td><td>18 Ohm</td><td>25 Ohm</td></tr><tr><td>0x0C00</td><td>50 Ohm</td><td>18 Ohm</td></tr><tr><td>0x0D00</td><td>35 Ohm</td><td>18 Ohm</td></tr><tr><td>0x0E00</td><td>25 Ohm</td><td>18 Ohm</td></tr><tr><td>0x0F00</td><td>18 Ohm</td><td>18 Ohm</td></tr></table></td></tr>
</table>

Methods

None

Appendix A - Miscellaneous

This section goes over miscellaneous information.

PyD3XX Constants

```
NULL = ctypes.c_void_p(0)

# FT_Status values.
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_NO_MORE_ITEMS = 18
FT_TIMEOUT = 19
FT_OPERATION_ABORTED = 20
FT_RESERVED_PIPE = 21
FT_INVALID_CONTROL_REQUEST_DIRECTION = 22
FT_INVALID_CONTROL_REQUEST_TYPE = 23
FT_IO_PENDING = 24
FT_IO_INCOMPLETE = 25
FT_HANDLE_EOF = 26
FT_BUSY = 27
FT_NO_SYSTEM_RESOURCES = 28
FT_DEVICE_LIST_NOT_READY = 29
FT_DEVICE_NOT_CONNECTED = 30
FT_INCORRECT_DEVICE_PATH = 31
FT_OTHER_ERROR = 32

# Describe dictionaries for easier error code conversion.
FT_STATUS_STR = {
    FT_OK: "FT_OK",
    FT_INVALID_HANDLE: "FT_INVALID_HANDLE",
    FT_DEVICE_NOT_FOUND: "FT_DEVICE_NOT_FOUND",
    FT_DEVICE_NOT_OPENED: "FT_DEVICE_NOT_OPENED",
    FT_IO_ERROR: "FT_IO_ERROR",
    FT_INSUFFICIENT_RESOURCES: "FT_INSUFFICIENT_RESOURCES",
    FT_INVALID_PARAMETER: "FT_INVALID_PARAMETER",
    FT_INVALID_BAUD_RATE: "FT_INVALID_BAUD_RATE",
    FT_DEVICE_NOT_OPENED_FOR_ERASE: "FT_DEVICE_NOT_OPENED_FOR_ERASE",
    FT_DEVICE_NOT_OPENED_FOR_WRITE: "FT_DEVICE_NOT_OPENED_FOR_WRITE",
    FT_FAILED_TO_WRITE_DEVICE: "FT_FAILED_TO_WRITE_DEVICE",
    FT_EEPROM_READ_FAILED: "FT_EEPROM_READ_FAILED",
    FT_EEPROM_WRITE_FAILED: "FT_EEPROM_WRITE_FAILED",
    FT_EEPROM_ERASE_FAILED: "FT_EEPROM_ERASE_FAILED",
    FT_EEPROM_NOT_PRESENT: "FT_EEPROM_NOT_PRESENT",
    FT_EEPROM_NOT_PROGRAMMED: "FT_EEPROM_NOT_PROGRAMMED",
    FT_INVALID_ARGS: "FT_INVALID_ARGS",
    FT_NOT_SUPPORTED: "FT_NOT_SUPPORTED",
    FT_NO_MORE_ITEMS: "FT_NO_MORE_ITEMS",
    FT_TIMEOUT: "FT_TIMEOUT",
    FT_OPERATION_ABORTED: "FT_OPERATION_ABORTED",
    FT_RESERVED_PIPE: "FT_RESERVED_PIPE",
```

```
    FT_INVALID_CONTROL_REQUEST_DIRECTION: "FT_INVALID_CONTROL_REQUEST_DIRECTION",
    FT_INVALID_CONTROL_REQUEST_TYPE: "FT_INVALID_CONTROL_REQUEST_TYPE",
    FT_IO_PENDING: "FT_IO_PENDING",
    FT_IO_INCOMPLETE: "FT_IO_INCOMPLETE",
    FT_HANDLE_EOF: "FT_HANDLE_EOF",
    FT_BUSY: "FT_BUSY",
    FT_NO_SYSTEM_RESOURCES: "FT_NO_SYSTEM_RESOURCES",
    FT_DEVICE_LIST_NOT_READY: "FT_DEVICE_LIST_NOT_READY",
    FT_DEVICE_NOT_CONNECTED: "FT_DEVICE_NOT_CONNECTED",
    FT_INCORRECT_DEVICE_PATH: "FT_INCORRECT_DEVICE_PATH",
    FT_OTHER_ERROR: "FT_OTHER_ERROR"
}
#^ FT_STATUS_STR[FT_OTHER_ERROR] == FT_STATUS_STR[32] == "FT_OTHER_ERROR"
FT_STATUS = {v: k for k, v in FT_STATUS_STR.items()}
#^ FT_STATUS["FT_OTHER_ERROR"] == FT_OTHER_ERROR == 32

FT_DEVICE_UNKNOWN = 3
FT_DEVICE_600 = 600
FT_DEVICE_601 = 601
FT_FLAGS_OPENED = 1
FT_FLAGS_HISPEED = 2
FT_FLAGS_SUPERSPEED = 4
FTPipeTypeControl = 0
FTPipeTypeIsochronous = 1
FTPipeTypeBulk = 2
FTPipeTypeInterrupt = 3
FT_LIST_NUMBER_ONLY = int("0x80000000", 16)
FT_LIST_BY_INDEX = int("0x40000000", 16)
FT_LIST_ALL = int("0x20000000", 16)
FT_OPEN_BY_SERIAL_NUMBER = int("0x00000001", 16)
FT_OPEN_BY_DESCRIPTION = int("0x00000002", 16)
FT_OPEN_BY_LOCATION = int("0x00000004", 16)
FT_OPEN_BY_GUID = int("0x00000008", 16)
FT_OPEN_BY_INDEX = int("0x00000010", 16)
FT_GPIO_DIRECTION_IN = 0
FT_GPIO_DIRECTION_OUT = 1
FT_GPIO_VALUE_LOW = 0
FT_GPIO_VALUE_HIGH = 1
FT_GPIO_0 = 0
FT_GPIO_1 = 1
E_FT_NOTIFICATION_CALLBACK_TYPE_DATA = 0
E_FT_NOTIFICATION_CALLBACK_TYPE_GPIO = 1
E_FT_NOTIFICATION_CALLBACK_TYPE_INTERRUPT = 2
```

```
CONFIGURATION_OPTIONAL_FEATURE_DISABLEALL = 0
CONFIGURATION_OPTIONAL_FEATURE_ENABLEBATTERYCHARGING = int("0x001", 16)
CONFIGURATION_OPTIONAL_FEATURE_DISABLECANCELSESSIONUNDERRUN = int("0x002",16)
CONFIGURATION_OPTIONAL_FEATURE_ENABLENOTIFICATIONMESSAGE_INCH1 = int("0x004", 16)
CONFIGURATION_OPTIONAL_FEATURE_ENABLENOTIFICATIONMESSAGE_INCH2 = int("0x008", 16)
CONFIGURATION_OPTIONAL_FEATURE_ENABLENOTIFICATIONMESSAGE_INCH3 = int("0x010", 16)
CONFIGURATION_OPTIONAL_FEATURE_ENABLENOTIFICATIONMESSAGE_INCH4 = int("0x020", 16)
CONFIGURATION_OPTIONAL_FEATURE_ENABLENOTIFICATIONMESSAGE_INCHALL = int("0x03C", 16)
CONFIGURATION_OPTIONAL_FEATURE_DISABLEUNDERRUN_INCH1 = int("0x040", 16)
CONFIGURATION_OPTIONAL_FEATURE_DISABLEUNDERRUN_INCH2 = int("0x080", 16)
CONFIGURATION_OPTIONAL_FEATURE_DISABLEUNDERRUN_INCH3 = int("0x100", 16)
CONFIGURATION_OPTIONAL_FEATURE_DISABLEUNDERRUN_INCH4 = int("0x200", 16)
CONFIGURATION_OPTIONAL_FEATURE_SUPPORT_ENABLE_FIFO_IN_SUSPEND = int("0x400", 16)
# available in RevB parts only
CONFIGURATION_OPTIONAL_FEATURE_SUPPORT_DISABLE_CHIP_POWERDOWN = int("0x800", 16)
# available in RevB parts only
CONFIGURATION_OPTIONAL_FEATURE_DISABLEUNDERRUN_INCHALL = int("0x3C0", 16)
CONFIGURATION_OPTIONAL_FEATURE_ENABLEALL = int("0xFFFF", 16)

FT_DEVICE_DESCRIPTOR_TYPE = int("0x01", 16)
FT_CONFIGURATION_DESCRIPTOR_TYPE = int("0x02", 16)
FT_STRING_DESCRIPTOR_TYPE = int("0x03", 16)
FT_INTERFACE_DESCRIPTOR_TYPE = int("0x04", 16)
FT_ENDPOINT_DESCRIPTOR_TYPE = int("0x05", 16) # THIS IS PYTHON API SPECIFIC.
```

Document References

D3XX Programmer's Guide (AN_379): Application Notes - FTDI
FT600Q-FT601Q SuperSpeed USB3.0 IC Datasheet: USB IC Data Sheets - FTDI

Appendix B – Version History

This section goes over version history of this document and PyD3XX.

Version	Date	Changes
1.0.0	February 5th, 2025	Initial release.
1.0.1	February 10th, 2025	Updated sections 2.7 & 2.8; parameters table and example code were modified to reflect Linux operation. Added paragraph at start of section 2 that goes over possible Platform string values.
1.0.2	February 12th, 2025	Updated Table 1.1.0 to reflect there's no WinUSB dynamic library for 32-bit systems. Added Table B.0.1 to track PyD3XX version history. Fixed some table formatting and alignment.
1.0.4	April 8th, 2025	PyD3XX Programmer's Guide document will now be released synchronously with PyD3XX updates and share the same version number. Section 1 was updated to reflect new driver library files packaged

		with PyD3XX. Sections 2.8, 2.10, 2.10B: Fixed descriptions where write was used when read was meant. Section 2.9B, 2.10B: Updated examples and descriptions to indicate overlapped object is required.
1.0.5	April 8th, 2025	No major change to document. PyD3XX was updated.
1.0.6	May 7th, 2025	Sections 2.38 & 2.39: Updated description for Device parameter to match function behavior. Section 2.14: Added note about StreamSize limitation for the WinUSB drivers. Updated example code + output.
1.0.7	July 30th, 2025	Table 1.2.0: Added pipe index column and renamed index column to FIFO index column. Section 2.14: Clarified StreamSize note and mentioned Full Speed. Section 3.2: Added note about Full Speed for the Flags attribute.
1.0.8	August 21st, 2025	Table 1.1.0: Updated D3XX WinUSB driver versions to 1.4.0.1.

Table B.0.0 – Document History

Version	Date	Changes
1.0.0	July 16th, 2024	Initial release.
1.0.1	July 16th, 2024	Initial release.
1.0.2	February 10th, 2025	Fixed bugs on macOS & Linux. Added GetDriverVersion() & GetLibraryVersion() functions. Added FT_ResetDevicePort() function.
1.0.3	February 12th, 2025	PyD3XX no longer loads the WinUSB dynamic library file for 32-bit systems as it is only for 64-bit systems. PyD3XX will now prioritize using the D3XX dynamic library over the WinUSB dynamic library if both drivers are installed.
1.0.4	April 8th, 2025	PyD3XX now prioritizes loading the WinUSB dynamic libraries from FTDI for Windows for all architectures. D3XX library files packaged with PyD3XX have been updated. PyD3XX now uses windll "stdcall" calling convention for Windows systems instead of cdll "cdecl" calling convention.
1.0.5	April 8th, 2025	Fixed bug on Linux & macOS with functions like FT_GetDeviceInfoList() not working correctly due to byte-misalignment by changing size of unsigned long and DWORD to match Windows.
1.0.6	May 7th, 2025	Fixed byte misalignment issue when getting MaximumPacketSize in FT_GetPipeInformation(). Added major error message to FT_SetStreamPipe() that's triggered when an invalid stream size is set.
1.0.7	July 30th, 2025	Fixed allocated buffer size of pipe information struct in FT_GetPipeInformation().
1.0.8	August 21st, 2025	Updated PyD3XX to not use local encoding when checking for drivers on start-up. Updated Windows WinUSB D3XX library files to version 1.4.0.1.

Table B.0.1 – PyD3XX History