Cypress CyUsb3.sys Programmer's Reference

Table of Contents

1 Driver Overview

The CYUSB3.SYS driver is licensed for distribution ONLY with Cypress USB products and products that employ Cypress USB chips.

CyUSB3.sys is a kernel mode USB function driver,it is capable of communicating with any USB 2.0 and USB3.0 compliant devices. The driver is general-purpose, understanding primitive USB commands, but not implementing higher-level, USB device-class specific commands. For this reason, the driver is not capable, for instance, of interfacing a USB mass storage class device to the Windows file system. Please note that this release does not include the USB3.0 bulk streaming interface.

The driver would be ideal for communicating with a vendor-specific device from a custom USB application or to send low-level USB requests to any USB device for experimental or diagnostic applications.

In order to use the driver to communicate with a device, Windows must match the device to the driver.

The class library, CyAPI.lib and Cyusb.dll, provides a high-level programming interface to the driver. This help file documents the low-level, more cumbersome and explicit programming interface.

The driver supports following operating systems and platform.

1. Windows XP 32 bit
2. Windows Vista 32/64bit
3. Windows 7 and 32/64 bit
4. Windows 8 Beta 32/64 bit

Cypress signed CyUSB3.sys driver for following Cypress VID/PID pairs.

1. VID_04B4&PID_00F0
2. VID_04B4&PID_00F1
3. VID_04B4&PID_00F3
4. VID_04B4&PID_4720

Please note, the driver is signed for above mentioned pairs of Cypress VID/PID. If you modify CyUSB3.inf to add your own VID/PID, the driver will become unsigned. However, Windows OS will allow to install the unsigned driver with warning messages on 32-bit and 64-bit platform. You can go for WHQL driver certification with your own VID/PID once your product is ready.

Features

- Windows Driver Foundation (WDF) compliant
- Compatible with any USB 2.0 compliant device
- Compatible with Cypress USB 3.0 compliant device
- Supports basic USB3.0 features.
- Supports Windows Plug and play and power management
- Supports USB Remote Wake-up
- Supports Control, Bulk, Interrupt and Isochronous transfers
- Supports multiple USB devices connected at once
- Supports customizable driver GUID without re-building the driver
- Supports high bandwidth data transfers passing multiple packets per micro-frame

2 Modifying CyUSB3.INF

The CYUSB3.INF file can be modified to accomplish several different objectives. These are:

1. Add a device's identifiers to the driver
2. Replace Cypress strings that are displayed during driver installation
3. Implement a custom GUID for the driver

NOTE: x86 refers to the 32bit OS and amd64 refers to 64 bit OS.

Add a device's identifiers to the driver

The following steps describe the process of adding a device's vendor ID and product ID to the CYSUB3. INF file.

1. Locate the following sections **[Device],[Device.NT],[Device.Ntx86]** and **[Device.Ntamd64]** and remove the semicolon of each item under the each section
   ;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUSB3, USB\VID_XXXX&PID_XXXX

2. Change the **VID_XXXX** to contain the hexadecimal value of the **VendorID** for the device and change the **PID_XXXX** to contain the hexadecimal value of the **ProductID** for the device

   For example, a device with vendorID **0x04B4** and productID **0xDE01** would have a new entry in the above listed sections like following
   %VID_04B4&PID_DE01.DeviceDesc%=CyUSB3, USB\VID_04B4&PID_DE01

3. Change [String] section for Device Description according to the Vendor ID and Product ID.
   VID_XXXX&PID_XXXX.DeviceDesc="Cypress USB3.0 Generic Driver"

4. Change the **VID_XXXX** to contain the hexadecimal value of the **VendorID** for the device

   Change the **PID_XXXX** to contain the hexadecimal value of the **ProductID** for the device

   For example, a device with vendorID **0x04B4** and productID **0xDE01** would have a new entry in the [Strings] section like the following
   VID_04B4&PID_DE01.DeviceDesc="Cypress FX3 Bulk sample"

Replace Cypress strings

If you plan to do more than just add your device's VID/PID to the CYUSB3.INF file, it is strongly recommended that you create your own .INF file and a copy of CYUSB3.SYS that you have re-named. The remaining instructions assume that you have created your own .INF file to match your newly named copy of CYUSB3.SYS.

The driver can be customized to report a company other than Cypress as its manufacturer and provider.

1. Locate the **[Strings]** section at the bottom of the CYUSB3.INF file.

2. Change the quoted **CYUSB3_Provider** string.

3. Change the quoted **CYUSB3_DisplayName** string.

4. Change the quoted **CYUSB3_Company** string.

5. Change the quoted **CYUSB3_Description** string.

Implement a custom GUID

Applications software usually accesses the driver using the driver's Global Unique IDentifier (GUID). Each driver in the Windows system should have a unique GUID. By employing distinct GUIDs, multiple instances of CYUSB3.SYS from different hardware vendors can exist on a given system without colliding.

1. To change the driver's GUID,

2. Use the GUIDGEN.EXE utility (distributed with Microsoft Visual Studio) to get a new GUID.

3. Locate the **[Strings]** section in the CyUSB3.inf file

4. Locate the line
   CYUSB3.GUID="{AE18AA60-7F6A-11d4-97DD-00010229B959}"
   and replace the quoted GUID string with the new one you created. (Retain the curly braces.)

Execute a script at start-up

The CYUSB3.SYS driver can be used to perform transfers to the default control endpoint (endpoint address 0) when the device is started.

To configure the driver to perform a control transfer at startup

1. Use the CyControl.exe application to create a script file containing the control transfer commands.

2. Save the script as a file named MyDevice.SPT

3. Place that script file in the same directory as the the driver's .INF file

A common use of this feature is to have the driver play a script which downloads a firmware image to the USB device, thereby modifying its "personality" and usually causing it to re-enumerate on the bus. If this re-enumeration occurs with the same VID/PID as the original "personality", the script will be executed again and again in an un-ending loop.

To avoid this endless loop scenario, the second personality should enumerate with a different VID/PID than the one which caused the script to play.

The .inf file can be modified to play a script when one VID/PID is enumerated and to simply load the driver when a different VID/PID is detected.

How to disable the CyScript feature

To disable this feature the user needs to delete the key 'DriverEXECSCRIPT' from the registry.
The following steps should be followed to delete the key.

1. Execute the 'regedit.exe' application.
2. Search for the 'DriverEXECSCRIPT'.
3. Delete the key 'DriverEXECSCRIPT'.
4. Close 'regedit.exe'.

The following is an excerpt from a .inf file that plays a script called MyDevice.spt when VID/PID of 04B4/00F3 is enumerated. If VID/PID 0547/00F0 enumerates, the script is not played.

NOTE: For FX3 devices, the MyDevice.spt script will play only when the connected device supports FX3 boot commands.

Sample CYUSB3.INF file using the CyScript feature with VID-0x04B4 and PID-0x00F3 and 0x00F0

```
; Installation INF for the Cypress Generic USB Driver for OS unknown
; Processor support for x86 based platforms.
;
; (c) Copyright 2012 Cypress Semiconductor Corporation
;

[Version]
Signature="$WINDOWS NT$"
Class=USB
ClassGUID={36FC9E60-C465-11CF-8056-444553540000}
provider=%CYUSB3_Provider%
CatalogFile=CYUSB3.cat
DriverVer=01/23/2012,1.0.0.01

[SourceDisksNames]
1=%CYUSB3_Install%,,,

[SourceDisksFiles]
CYUSB3.sys = 1

[DestinationDirs]
CYUSB3.Files.Ext = 10,System32\Drivers
MyDevice.Files.Ext = 10,System32\MyDevice

[ControlFlags]
ExcludeFromSelect = *

[Manufacturer]
%CYUSB3_Provider%=Device,NT,NTx86,NTamd64

;for all platforms
[Device.NT]
%VID_04B4&PID_00F3.DeviceDesc%=MyDevice, USB\VID_04B4&PID_00F3
%VID_04B4&PID_00F0.DeviceDesc%=CYUSB3, USB\VID_04B4&PID_00F0

;for x86 platforms
[Device.NTx86]
%VID_04B4&PID_00F3.DeviceDesc%=MyDevice, USB\VID_04B4&PID_00F3
%VID_04B4&PID_00F0.DeviceDesc%=CYUSB3, USB\VID_04B4&PID_00F0
```

```
;for x64 platforms
[Device.NTamd64]
%VID_04B4&PID_00F3.DeviceDesc%=MyDevice, USB\VID_04B4&PID_00F3
%VID_04B4&PID_00F0.DeviceDesc%=CYUSB3, USB\VID_04B4&PID_00F0

[MyDevice]
CopyFiles=CYUSB3.Files.Ext,MyDevice.Files.Ext
AddReg=CYUSB3.AddReg

[MyDevice.HW]
AddReg=MyDevice.AddReg.Guid

[MyDevice.Services]
Addservice = CYUSB3,2,CYUSB3.AddService

[MyDevice.NT]
CopyFiles=CYUSB3.Files.Ext, MyDevice.Files.Ext
AddReg=CYUSB3.AddReg

[MyDevice.NT.HW]
AddReg=MyDevice.AddReg.Guid

[MyDevice.NT.Services]
Addservice = CYUSB3,2,CYUSB3.AddService

[MyDevice.NTx86]
CopyFiles=CYUSB3.Files.Ext, MyDevice.Files.Ext
AddReg=CYUSB3.AddReg

[MyDevice.NTx86.HW]
AddReg=MyDevice.AddReg.Guid

[MyDevice.NTx86.Services]
Addservice = CYUSB3,2,CYUSB3.AddService

[MyDevice.NTamd64]
CopyFiles=CYUSB3.Files.Ext, MyDevice.Files.Ext
AddReg=CYUSB3.AddReg

[MyDevice.NTamd64.HW]
AddReg=MyDevice.AddReg.Guid

[MyDevice.NTamd64.Services]
Addservice = CYUSB3,2,CYUSB3.AddService

[MyDevice.AddReg.Guid]
HKR,,DriverGUID,,%CYUSB3.GUID%
HKR,,DriverEXECSCRIPT,,%MyDevice.EXECSCRIPT%

[MyDevice.Files.Ext]
MyDevice.spt

[CYUSB3.NT]
CopyFiles=CYUSB3.Files.Ext
AddReg=CYUSB3.AddReg

[CYUSB3.NT.HW]
```

```
AddReg=CYUSB3.AddReg.Guid

[CYUSB3.NT.Services]
Addservice = CYUSB3,2,CYUSB3.AddService


[CYUSB3.NTx86]
CopyFiles=CYUSB3.Files.Ext
AddReg=CYUSB3.AddReg

[CYUSB3.NTx86.HW]
AddReg=CYUSB3.AddReg.Guid

[CYUSB3.NTx86.Services]
Addservice = CYUSB3,2,CYUSB3.AddService

[CYUSB3.NTamd64]
CopyFiles=CYUSB3.Files.Ext
AddReg=CYUSB3.AddReg

[CYUSB3.NTamd64.HW]
AddReg=CYUSB3.AddReg.Guid

[CYUSB3.NTamd64.Services]
Addservice = CYUSB3,2,CYUSB3.AddService


[CYUSB3.AddReg]
; Deprecating - do not use in new apps to identify a CYUSB3 driver
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,CYUSB3.sys
; You may optionally include a check for DriverBase in your application to check for a CYUSB3 driver
HKR,,DriverBase,,CYUSB3.sys
HKR,"Parameters","MaximumTransferSize",0x10001,4096
HKR,"Parameters","DebugLevel",0x10001,2
HKR,,FriendlyName,,%CYUSB3_Description%

[CYUSB3.AddService]
DisplayName    = %CYUSB3_Description%
ServiceType    = 1                  ; SERVICE_KERNEL_DRIVER
StartType      = 3                  ; SERVICE_DEMAND_START
ErrorControl   = 1                  ; SERVICE_ERROR_NORMAL
ServiceBinary  = %10%\System32\Drivers\CYUSB3.sys
AddReg         = CYUSB3.AddReg
LoadOrderGroup = Base

[CYUSB3.Files.Ext]
CYUSB3.sys

[CYUSB3.AddReg.Guid]
HKR,,DriverGUID,,%CYUSB3.GUID%

;---------- WDF Coinstaller installation
[SourceDisksFiles]
WdfCoInstaller01009.dll=1 ; make sure the number matches with SourceDisksNames

[DestinationDirs]
CoInstaller_CopyFiles = 11
```

```
[CYUSB3.NTamd64.CoInstallers]
AddReg=CoInstaller_AddReg
CopyFiles=CoInstaller_CopyFiles

[CYUSB3.NTx86.CoInstallers]
AddReg=CoInstaller_AddReg
CopyFiles=CoInstaller_CopyFiles

[CoInstaller_CopyFiles]
WdfCoInstaller01009.dll

[CoInstaller_AddReg]
HKR,,CoInstallers32,0x00010000, "WdfCoInstaller01009.dll,WdfCoInstaller"

[CYUSB3.NTamd64.Wdf]
KmdfService = CYUSB3, CYUSB3_wdfsect

[CYUSB3.NTx86.Wdf]
KmdfService = CYUSB3, CYUSB3_wdfsect

[CYUSB3_wdfsect]
KmdfLibraryVersion = 1.9


[Strings]
CYUSB3_Provider    = "Cypress"
CYUSB3_Company     = "Cypress Semiconductor Corporation"
CYUSB3_Description = "Cypress Generic USB3.0 Driver"
CYUSB3_DisplayName = "Cypress USB3.0 Generic"
CYUSB3_Install     = "Cypress CYUSB3.0 Driver Installation Disk"
VID_04B4&PID_00F3.DeviceDesc="Cypress USB BootLoader"
VID_04B4&PID_00F0.DeviceDesc="Cypress BULK LOOP"
CYUSB3.GUID="{AE18AA60-7F6A-11d4-97DD-00010229B959}"
CYUSB3_Unused      = "."
MyDevice.EXECSCRIPT="\systemroot\system32\MyDevice\MyDevice.spt"
```

3 Matching Devices to the Driver

Usually matching of a USB device to the CYUSB3.SYS driver will need to be manually configured.

Following are the steps user has to follow to install driver on Windows OS.

Step A : Add the device's VendorID and ProductID to the CYUSB3.INF file.
Step B : Force Windows to use the CYUSB3.SYS driver with the device.

Though similar, these steps are slightly different for WinXP and Windows Vista and 7

3.1 Windows XP

Usually, matching of a USB device to the CYUSB3.SYS driver will need to be manually configured.

Please follow below steps to update the INF file and driver installation on Windows XP.

Note: Please skip step B to install driver on 32-bit OS.

Step A : Please follow the below steps to add the device's VendorID and ProductID to the CYUSB3.INF file.

1. After installation of the Cypress Suite USB installer, the driver file is located in a Driver subdirectory of the install directory. (Default is C:\Program Files\Cypress\FX3 Host Software\Driver\bin.).

2. Open the file CYUSB3.INF with a text editor (notepad.exe, for instance)

3. Locate the following sections [Device],[Device.NT],[Device.Ntx86] and [Device.Ntamd64] and remove the semicolon of each item under the each section
   ;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb3, USB\VID_XXXX&PID_XXXX.

4. Change the VID_XXXX to contain the hexadecimal value of the VendorID for the device and change the PID_XXXX to contain the hexadecimal value of the ProductID for the device.

   For example, a device with vendorID 0x04B4 and productID 0xDE01 would have a new entry in the above listed sections like following:
   %VID_04B4&PID_DE01.DeviceDesc%=CyUSB3, USB\VID_04B4&PID_DE01

5. Change [String] section for Device Description according to the Vendor ID and Product ID.
   VID_XXXX&PID_XXXX.DeviceDesc="Cypress USB3.0 Generic Driver"

6. Change the VID_XXXX to contain the hexadecimal value of the VendorID for the device and change the PID_XXXX to contain the hexadecimal value of the ProductID for the device.

   For example, a device with vendorID 0x04B4 and productID 0xDE01 would have a new entry in the [Strings] section like the following:
   VID_04B4&PID_DE01.DeviceDesc="Cypress FX3 Bulk loopback"

7. Save the file.

Step B : Please follow the below steps to force WindowsXP to use the cyusb3.sys driver with the device.

1. Connect the device to the PC

2. If Windows prompts for a driver or indicates that it needs a driver, direct the PC to use the CYUSB3.SYS driver by steering it to the CYUSB3.INF file in the [InstallDir]\Driver directory.

3. If Windows does not prompt for a driver, it has already matched the device to a driver itself. In this case, you will need to see if the CYUSB3.SYS driver was selected and, if not, manually instruct Windows to use that driver.

4. Right-click My Computer and select the Manage menu item.

5. In the Computer Management window, select Device Manager
6. In the right window pane, click the + icon next to Universal Serial Bus controllers
7. Locate your device in the list and double click on it
8. Select the Driver tab in the Properties dialog that comes up
9. Click on the Driver Details button.
10. If the displayed driver file is CYUSB3.SYS, Windows has already matched the device to this driver and you should click OK and Cancel . If not, proceed with the remaining steps.
11. Click OK
12. Click Update Driver
13. Select Install from a list or specific location (Advanced)
14. Click Next
15. Select Don't search. I will choose the driver to install.
16. Click Next
17. Click Have Disk
18. Click Browse
19. Navigate to the directory containing CYUSB3.SYS (wxp(Windows XP) and select x86(32-bit OS) or x64(64-bit OS)) based on the platform you want to install driver on.
20. CYUSB3.INF should be automatically placed in the File name field
21. Click Open
22. Click OK
23. Click Next
24. It will popup message saying Unsigned driver, Please select 'Install driver software anyway' and click ok.
25. Click Finish
26. Click Close
27. Don't re-boot your system if Windows suggests that you must. You may need to unplug and re-plug your device, however.

3.2 Windows Vista ,Win7 and Win8 Beta

Usually, matching of a USB device to the CYUSB3.SYS driver will need to be manually configured.

Please follow below steps to update the INF file and driver installation on Windows Vista, Windows 7 and Windows 8 Beta.

Note: Please skip step B to install driver on 32-bit OS. To install driver on Windows 8 Beta , please use the \bin\Win7\ directory.

Step A : Please follow the below steps to add the device's VendorID and ProductID to the CYUSB3.INF file.

1. After installation of the Cypress Suite USB installer, the driver file is located in a Driver subdirectory of the install directory. (Default is C:\Cypress\Cypress USBSuite\driver\bin\)

2. Open the file CYUSB3.INF with a text editor (notepad.exe, for instance)

3. Locate the following sections [Device],[Device.NT],[Device.Ntx86] and [Device.Ntamd64] and remove the semicolon of each item under the each section
   ;%VID_XXXX&PID_XXXX.DeviceDesc%=CyUsb3, USB\VID_XXXX&PID_XXXX

4. Change the VID_XXXX to contain the hexadecimal value of the VendorID for the device and change the PID_XXXX to contain the hexadecimal value of the ProductID for the device

   For example, a device with vendorID 0x04B4 and productID 0xDE01 would have a new entry in the above listed sections like following:
   %VID_04B4&PID_DE01.DeviceDesc%=CyUSB3, USB\VID_04B4&PID_DE01

5. Change [String] section for Device Description according to the Vendor ID and Product ID.
   VID_XXXX&PID_XXXX.DeviceDesc="Cypress USB3.0 Generic Driver)"

6. Change the VID_XXXX to contain the hexadecimal value of the VendorID for the device and change the PID_XXXX to contain the hexadecimal value of the ProductID for the device

   For example, a device with vendorID 0x04B4 and productID 0xDE01 would have a new entry in the [Strings] section like the following:
   VID_04B4&PID_DE01.DeviceDesc="Cypress FX3 Bulk sample"

7. Save the file.

Step B : Force Windows Vista/Windows 7 to use the cyusb3.sys driver with the device.

1. Connect the device to the PC

2. If Windows prompts for a driver or indicates that it needs a driver, direct the PC to use the CYUSB3.SYS driver by steering it to the CYUSB3.INF file in the [InstallDir]\driver directory.

3. If Windows does not prompt for a driver, it has already matched the device to a driver itself. In this case, you will need to see if the CYUSB3.SYS driver was selected and, if not, manually instruct Windows to use that driver.

4. Right-click My Computer and select the Manage menu item.

5. In the Computer Management window, select Device Manager
6. In the right window pane, click the + icon next to Universal Serial Bus controllers
7. Locate your device in the list and double click on it
8. Select the Driver tab in the Properties dialog that comes up
9. Click on the Driver Details button.
10. If the displayed driver file is CYUSB3.SYS, Windows has already matched the device to this driver and you should click OK and Cancel . If not, proceed with the remaining steps.
11. Click OK
12. Click Update Driver
13. Select Browse my computer for driver software
14. Click Next
15. Select Let me pick from a list of device drivers on my computer
16. Click Next
17. Select Select your device's type from list below and Select show all device
18. Click Next
19. Click Have Disk
20. Click Browse
21. Navigate to the directory containing CYUSB3.SYS (Directory Name for various Operating System and platform: wlh(windows Vista) and win7(windows 7 and Windows 8 Beta) and select x86(32-bit OS) or x64(64-bit OS)) based on the platform you want to install driver on.
22. CYUSB3.INF should be automatically placed in the File name field
23. Click Open
24. Click OK
25. Click Next
26. It will popup message saying Unsigned driver, Please select 'Install driver software anyway' and click ok.
27. Click Finish
28. Click Close

29. Don't re-boot your system if Windows suggests that you must. You may need to unplug and re-plug your device, however.

4 Reinstalling the Driver

While reinstalling the driver with another .inf file which contains the same VID- PID combination, it's safe to remove all oemXX.inf and oemXX.pnf files from the directory "C:\WINDOWS\inf\" which have same VID-PID combination.

Note:
Installing the driver using .inf file, Windows creates corresponding oemXX.inf and oemXX.pnf backup files in the directory "C:\WINDOWS\inf\". There is a chance for mistaking the backup .inf file instead of the new .inf file that customer really wants to install.

5 The IOCTL Interface

Applications software communicates with the CYUSB3.SYS driver primarily through the DeviceIoControl ( ) function. (See the Windows SDK documentation for details about DeviceIoControl.)

Calls to DeviceIoControl require an IO Control (aka IOCTL) code parameter. The IOCTL codes define the programming interface that a driver supports and are particular to any given driver. The control code specified in a DeviceIoControl( ) call determines the values that must be specified for the other DeviceIoControl parameters.

This help file provides the IOCTL 'dictionary' for the CYUSB3.SYS driver.

Example

```
DWORD dwBytes = 0;
UCHAR EndptAddress = 0x82;

DeviceIoControl( hDevice, IOCTL_ADAPT_RESET_PIPE,
                 &EndptAddress, sizeof ( EndptAddress),
                 NULL, 0,
                 &dwBytes, NULL);
```

5.1 Getting a Handle to the Driver

In order to use the IOCTL codes supported by the driver, you will need to obtain a Windows handle to the driver.

A very simple way to accomplish this is to utilize the CyAPI class library. After creating a CCyUSBDevice object, a handle to the driver will have been setup automatically. Closing or deleting the CCyUSBDevice object frees the handle.

Example 1:

```
CCyUSBDevice *USBDevice = new CCyUSBDevice();
HANDLE hDevice = USBDevice->DeviceHandle();
.
.
.
.
delete USBDevice;
```

The more typical (and complex) way to obtain a handle is to make a sequence of SetupDi calls, passing the driver GUID declared in CyAPI.h. The default driver guid is defined as:

```
// {AE18AA60-7F6A-11d4-97DD-00010229B959}
static GUID CYUSBDRV_GUID = {0xae18aa60, 0x7f6a, 0x11d4, 0x97, 0xdd, 0x0, 0x1, 0x2,
0x29, 0xb9, 0x59};

The CyAPI library uses the following code to obtain a handle, using the GUID.
```

Example 2:

```
SP_DEVINFO_DATA devInfoData;
SP_DEVICE_INTERFACE_DATA  devInterfaceData;
PSP_INTERFACE_DEVICE_DETAIL_DATA functionClassDeviceData;

ULONG requiredLength = 0;
int deviceNumber = 0; // Can be other values if more than 1 device connected to driver

HDEVINFO hwDeviceInfo = SetupDiGetClassDevs ( (LPGUID) &CYUSBDRV_GUID,
                                              NULL,
                                              NULL,
                                              DIGCF_PRESENT| DIGCF_INTERFACEDEVICE);

if (hwDeviceInfo != INVALID_HANDLE_VALUE) {

devInterfaceData.cbSize = sizeof(devInterfaceData);

if (SetupDiEnumDeviceInterfaces ( hwDeviceInfo, 0, (LPGUID) &CYUSBDRV_GUID,
                                  deviceNumber, &devInterfaceData)) {

  SetupDiGetInterfaceDeviceDetail ( hwDeviceInfo, &devInterfaceData, NULL, 0,
```

```
                                        &requiredLength, NULL);

  ULONG predictedLength = requiredLength;

  functionClassDeviceData = (PSP_INTERFACE_DEVICE_DETAIL_DATA) malloc
(predictedLength);
  functionClassDeviceData->cbSize = sizeof (SP_INTERFACE_DEVICE_DETAIL_DATA);

  devInfoData.cbSize = sizeof(devInfoData);

if (SetupDiGetInterfaceDeviceDetail (hwDeviceInfo,
                                     &devInterfaceData,
                                     functionClassDeviceData,
                                     predictedLength,
                                     &requiredLength,
                                     &devInfoData)) {

    hDevice = CreateFile (functionClassDeviceData->DevicePath,
                          GENERIC_WRITE | GENERIC_READ,
                          FILE_SHARE_WRITE | FILE_SHARE_READ,
                          NULL,
                          OPEN_EXISTING,
                          FILE_FLAG_OVERLAPPED,
                          NULL);

    free(functionClassDeviceData);
    SetupDiDestroyDeviceInfoList(hwDeviceInfo);
  }
```

5.2 IOCTL_ADAPT_ABORT_PIPE

Description

This command is used to cancel pending IO requests on an endpoint.

A pointer to a variable containing the endpoint address is passed as the lpInBuffer parameter to the DeviceIoControl( ) function. A null pointer is passed as the lpOutBuffer parameter.

Example

```
DWORD dwBytes = 0;
UCHAR Address = 0x82;

DeviceIoControl( hDevice, IOCTL_ADAPT_ABORT_PIPE,
             &Address, sizeof ( UCHAR),
             NULL, 0,
             &dwBytes, NULL);
```

5.3 IOCTL_ADAPT_CYCLE_PORT

Description

This command power-cycles the USB port to which a specified device is attached., Power-cycling a port causes the device to be surprise-removed and re-enumerated.

NULL pointers are passed to DeviceIoControl in the pInBuffer and pOutBuffer parameters.

Example

```
DWORD dwBytes = 0;

DeviceIoControl( hDevice, IOCTL_ADAPT_CYCLE_PORT,
              NULL, 0,
              NULL, 0,
              &dwBytes, NULL);
```

5.4 IOCTL_ADAPT_GET_ADDRESS

Description

This command retrieves the USB address of the device from the Windows host controller driver.

A pointer to a 1-byte variable is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The size of the variable (1) is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
UCHAR DevAddr;

DeviceIoControl( hDevice, IOCTL_ADAPT_GET_ADDRESS,
             &DevAddr, sizeof ( UCHAR),
             &DevAddr, sizeof ( UCHAR),
             &dwBytes, NULL);
```

5.5 IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING

Description

This command retrieves the alternate interface setting for a particular interface of the attached device.

A pointer to a byte indicating the interface number is passed as the lpInBuffer parameter to the DeviceIoControl( ) function.

A pointer to a byte into which the alternate interface setting will be reported is passed as the lpOutBuffer parameter to the DeviceIoControl( ) function.

The length of the variables (1) is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
UCHAR intfc = 0;
UCHAR alt;

DeviceIoControl(hDevice, IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING,
            &intfc, sizeof (alt),
            &alt, sizeof (alt),
            &dwBytes, NULL);
```

5.6 IOCTL_ADAPT_GET_CURRENT_FRAME

Description

This command returns the current frame number from the host controller driver.

A pointer to a 4-byte variable is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The size of the variable (4) is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
ULONG CurrentFrame;
DeviceIoControl( hDevice, IOCTL_ADAPT_GET_CURRENT_FRAME,
                 DeviceIoControl( hDevice, IOCTL_ADAPT_GET_CURRENT_FRAME,
                 &CurrentFrame, sizeof ( ULONG),
                 &CurrentFrame, sizeof ( ULONG),
                 &dwBytes, NULL);
```

5.7 IOCTL_ADAPT_GET_DEVICE_NAME

Description

This command retrieves the Product string descriptor value for the attached device.

A pointer to a character buffer is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The length of the buffer is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
ULONG len = 256;
UCHAR *buf = new UCHAR[len];

DeviceIoControl(hDevice, IOCTL_ADAPT_GET_DEVICE_NAME,
             buf, len,
             buf, len,
             &dwBytes, NULL);

delete[] buf;
```

5.8 IOCTL_ADAPT_GET_DEVICE_POWER_STATE

Description

This IOCTL is no longer supported. It is available to keep backward compatibility with older interface library and application.

Microsoft WDF driver framework manages device power state internally.

5.9 IOCTL_ADAPT_GET_DEVICE_SPEED

Description

This command attempts to report the current operating speed of the USB device. It uses the **IsDeviceHighSpeed** routine, but this routine is only supported in Version 1 of the USBD interface. Windows 2K SP4, Windows XP and later all support Version 1 of the USBD interface. If the **IsDeviceHighSpeed** routine is not available, **DEVICE_SPEED_UNKNOWN** is returned. The possible return value of this IOCTL is defined in the cyioctl.h header file.

A pointer to a 4-byte variable is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The size of the variable (4) is passed in the nInBufferSize and nOutBufferSize parameters.

Defines (cyioctl.h)

```
#define DEVICE_SPEED_UNKNOWN       0x00000000
#define DEVICE_SPEED_LOW_FULL      0x00000001
#define DEVICE_SPEED_HIGH          0x00000002
#define DEVICE_SPEED_SUPER         0x00000004
```

Example

```
DWORD dwBytes = 0;
ULONG DevSpeed;
DeviceIoControl( hDevice, IOCTL_ADAPT_GET_DEVICE_SPEED,
           &DevSpeed, sizeof ( ULONG),
           &DevSpeed, sizeof ( ULONG),
           &dwBytes, NULL);
```

5.10 IOCTL_ADAPT_GET_DRIVER_VERSION

Description

This command retrieves the version of the driver.

A pointer to a 4-byte variable is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The size of the variable (4) is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
ULONG ver;

DeviceIoControl( hDevice, IOCTL_ADAPT_GET_DRIVER_VERSION,
             &ver, sizeof (ver),
             &ver, sizeof (ver),
             &dwBytes, NULL);
```

5.11 IOCTL_ADAPT_GET_FRIENDLY_NAME

Description

This command retrieves the string associated with the device in the [Strings] section of the CyUSB3.inf file.

A pointer to an array of unsigned characters is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The size of the array is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
PUCHAR FriendlyName = new UCHAR[256];

DeviceIoControl(hDevice, IOCTL_ADAPT_GET_FRIENDLY_NAME,
             FriendlyName, 256,
             FriendlyName, 256,
             &dwBytes, NULL);

delete[] FriendlyName;
```

5.12 IOCTL_ADAPT_GET_NUMBER_ENDPOINTS

Description

This command retrieves the number of endpoints enumerated by the current interface / alternate interface setting.

A null pointer is passed as the lpInBuffer parameter to the DeviceIoControl( ) function. Zero is passed as the nInBufferSize parameter.

The address of an unsigned character is passed as the lpOutBuffer parameter to the DeviceIoControl( ) function. The size of the variable (1) is passed in the nOutBufferSize parameter.

Example

```
DWORD dwBytes = 0;
UCHAR endPts;

DeviceIoControl( hDevice, IOCTL_ADAPT_GET_NUMBER_ENDPOINTS,
             NULL, 0,
             &endPts, sizeof (endPts),
             &dwBytes, NULL);
```

5.13 IOCTL_ADAPT_GET_TRANSFER_SIZE

Description

This IOCTL is no longer supported. It is available to keep backward compatibility with older interface library and application.

For more information on USB transfer size please refer link from Microsoft : http://msdn.microsoft.com/en-us/library/ff538112.aspx

Following are the maximum transfer size limits set into the CyUSB3.sys driver for various transfers.

1. Bulk and Interrupt Transfer
   4MBytes
2. Full Speed Isochronous Transfer
   256 Frames
3. High Speed and Super Speed Isochronous Transfer
   1024 Frames

5.14 IOCTL_ADAPT_GET_USBDI_VERSION

Description

This command retrieves the version of the USB Host Controller Driver in BCD format.

A pointer to a 4-byte variable is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The size of the variable (4) is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
ULONG ver;

DeviceIoControl( hDevice, IOCTL_ADAPT_GET_USBDI_VERSION,
             &ver, sizeof ( ver),
             &ver, sizeof ( ver),
             &dwBytes, NULL);
```

5.15 IOCTL_ADAPT_RESET_PARENT_PORT

Description

This command resets the upstream port of the device it manages. After a successful reset, the bus driver reselects the configuration and any alternative interface settings that the device had before the reset occurred. All pipe handles, configuration handles and interface handles remain valid.

A null pointer is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

Example

```
DWORD dwBytes;

DeviceIoControl( hDevice, IOCTL_ADAPT_RESET_PARENT_PORT,
                 NULL, 0,
                 NULL, 0,
                 &dwBytes, NULL);
```

5.16 IOCTL_ADAPT_RESET_PIPE

Description

This command resets an endpoint of the device, clearing any error or stall conditions on that endpoint. Pending data transfers are not cancelled by this command.

The address of a single byte is passed as the lpInBuffer parameter to the DeviceIoControl( ) function.

A null pointer is passed as the lpOutBuffer parameter.

Example

```
DWORD dwBytes;
UCHAR Address = 0x82;

DeviceIoControl( hDevice, IOCTL_ADAPT_RESET_PIPE,
             &Address, sizeof ( Address)
             NULL, 0,
             &dwBytes, NULL);
```

5.17 IOCTL_ADAPT_SELECT_INTERFACE

Description

This command sets the alternate interface setting for the primary interface of the attached device.

A pointer to a byte indicating the alternate interface setting is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl( ) function.

The length of the variable (1) is passed in the nInBufferSize and nOutBufferSize parameters.

Example

```
DWORD dwBytes = 0;
UCHAR alt = 2;

DeviceIoControl (hDevice, IOCTL_ADAPT_SELECT_INTERFACE,
             &alt, sizeof (alt),
             &alt, sizeof (alt),
             &dwBytes, NULL);
```

5.18 IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER

IOCTL_ADAPT_SEND_EP0_CONTROL _TRANSFER

Previous Top Next

Description

This command sends a control request to the default Control endpoint, endpoint zero.

DeviceIoControl( ) is passed a pointer to a two-part structure as both the lpInBuffer and lpOutBuffer parameters. This two-part structure contains a SINGLE_TRANSFER structure followed by a data buffer.

The SINGLE_TRANSFER structure contains all the parameters for the control request.

The buffer contains the transfer data.

NOTE : Please note that this IOCTL return device configuration inclusive of both interface for USB3.0 composite device. This is the limitation due to the USBDI bus interface. The USBDI doesn't support USB3.0 specific device configuration.

Example

```
union  {
struct   {
UCHAR Recipient: 5;
UCHAR  Type: 2;
UCHAR Direction: 1;
}  bmRequest;

UCHAR bmReq;
};

bmRequest. Recipient = 0; //  Device
bmRequest. Type      = 2; //  Vendor
bmRequest. Direction = 1; // IN command ( from  Device to Host)

int iXmitBufSize = sizeof( SINGLE_TRANSFER)  + bufLen; //  The size of the two-part
structure
UCHAR *pXmitBuf = new UCHAR[ iXmitBufSize];           //  Allocate the memory
ZeroMemory( pXmitBuf, iXmitBufSize);

PSINGLE_TRANSFER pTransfer = ( PSINGLE_TRANSFER) pXmitBuf; //  The SINGLE_TRANSFER comes
first
pTransfer->SetupPacket. bmRequest = bmReq;
pTransfer->SetupPacket. bRequest = ReqCode;
pTransfer->SetupPacket. wValue = Value;
pTransfer->SetupPacket. wIndex = Index;
pTransfer->SetupPacket. wLength = bufLen;
pTransfer->SetupPacket. ulTimeOut = TimeOut / 1000;
pTransfer->Reserved = 0;
pTransfer->ucEndpointAddress = 0x00;     //  Control pipe
pTransfer->IsoPacketLength = 0;
pTransfer->BufferOffset = sizeof ( SINGLE_TRANSFER);
pTransfer->BufferLength = bufLen;
```

```
DWORD dwReturnBytes;

DeviceIoControl (hDevice, IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER,
             pXmitBuf,  iXmitBufSize,
             pXmitBuf,  iXmitBufSize,
             &dwReturnBytes,  NULL);

// Copy data into  buf
UCHAR *ptr = pXmitBuf + sizeof (SINGLE_TRANSFER);
memcpy( buf, ptr, dwReturnBytes);
```

5.19 IOCTL_ADAPT_SEND_NON_EP0_TRANSFER

Description

This IOCTL command is used to request Bulk, Interrupt or Isochronous data transfers across corresponding USB device endpoints.

Regardless of whether the endpoint is an IN or an OUT endpoint, a pointer to a single data structure is passed to DeviceIoControl( ) as both the lpInBuffer and lpOutBuffer parameters. The driver expects that the pointer references a SINGLE_TRANSFER structure, followed by a data buffer. In the case of OUT endpoints, the buffer is expected to contain the data bytes to be transmitted. In the case of an IN endpoint, the buffer is expected to be the writeable memory for received data bytes.

Special ISOC Constraints

The endpoint maximum transfer size and buffer length parameter must both be a multiple of the endpoint's MaxPacketSize.

For ISOC transfers on a device operating at High speed or Super Speed, the following constraints apply to this command:

1) The buffer length parameter (bufLen in the below examples) must also be a multiple of the endpoint's MaxPacketSize * 8. Please also note that last packet is allow to send with partial size(less than Max packet) for both super and high speed Isochronous transfer.

2) For Super speed Isochronous endpoint only , if device define the MaxBurst in the super speed endpoint companion descriptor then Maxburst should be used to calculate the packet size= (MaxpacketSize * (MaxBurst+1). This packet length data will be sent over one micro frame interval.

Example

```
PUCHAR CCyBulkEndPoint::BeginDataXfer(PCHAR buf, LONG bufLen, OVERLAPPED *ov)
{
if (hDevice == INVALID_HANDLE_VALUE) return NULL;
int iXmitBufSize = sizeof (SINGLE_TRANSFER) + bufLen;
PUCHAR pXmitBuf = new UCHAR[iXmitBufSize];
ZeroMemory(pXmitBuf, iXmitBufSize);

PSINGLE_TRANSFER pTransfer = (PSINGLE_TRANSFER)pXmitBuf;
pTransfer->Reserved = 0;
pTransfer->ucEndpointAddress = Address;
pTransfer->IsoPacketLength = 0;
pTransfer->BufferOffset = sizeof (SINGLE_TRANSFER);
pTransfer->BufferLength = bufLen;
// Copy buf  into pXmitBuf
UCHAR *ptr = (PUCHAR) pTransfer + pTransfer->BufferOffset;
memcpy(ptr, buf, bufLen);
DWORD dwReturnBytes;
DeviceIoControl(hDevice, IOCTL_ADAPT_SEND_NON_EP0_TRANSFER,
                pXmitBuf, iXmitBufSize,
                pXmitBuf, iXmitBufSize,
                &dwReturnBytes, ov);
return  pXmitBuf;
```

```
}
```

5.20 IOCTL_ADAPT_SEND_NON_EP0_DIRECT

Description

This IOCTL is used to request Bulk, Interrupt or Isochronous data transfers across corresponding USB device endpoints.

The DeviceIoControl call requires two buffer parameters. For this command, the first buffer must contain a properly initialized SINGLE_TRANSFER structure.

The SINGLE_TRANSFER fields of BufferOffset and BufferLength should be set to 0 for this command.

The second buffer is for the actual transfer data. For an OUT endpoint, this will contain the data headed to the USB device. For an IN endpoint, this buffer will hold the data that is received from the device.

Special ISOC Constraints

The endpoint maximum transfer size and buffer length parameter must both be a multiple of the endpoint's MaxPacketSize.

For ISOC transfers on a device operating at High speed or Super Speed, the following constraints apply to this command:

1) The buffer length parameter (bufLen in the below examples) must also be a multiple of the endpoint's MaxPacketSize * 8. Please also note that last packet is allow to send with partial size(less than Max packet) for both super and high speed Isochronous transfer.

2) For Super speed Isochronous endpoint only , if device define the MaxBurst in the super speed endpoint companion descriptor then Maxburst should be used to calculate the packet size= (MaxpacketSize * (MaxBurst+1)). This packet length data will be sent over one micro frame interval.

The SINGLE_TRANSFER structure must be followed by additional space sufficient to hold the PACKET_INFO structures for the transfer (see examples #2 and #3, below).

Example #1 (Bulk and Interrupt endpoints)

```
PUCHAR CCyUSBEndPoint::BeginDirectXfer(PUCHAR buf, LONG bufLen, OVERLAPPED *ov)
{

if ( hDevice == INVALID_HANDLE_VALUE ) return NULL;
int iXmitBufSize = sizeof (SINGLE_TRANSFER);
PUCHAR pXmitBuf = new UCHAR[iXmitBufSize];
ZeroMemory (pXmitBuf, iXmitBufSize);

PSINGLE_TRANSFER pTransfer = (PSINGLE_TRANSFER) pXmitBuf;
pTransfer->ucEndpointAddress = Address;
pTransfer->IsoPacketLength = 0;
pTransfer->BufferOffset = 0;
pTransfer->BufferLength = 0;
DWORD dwReturnBytes;
DeviceIoControl (hDevice,
```

```
      IOCTL_ADAPT_SEND_NON_EP0_DIRECT,
      pXmitBuf, iXmitBufSize,
      buf, bufLen,
      &dwReturnBytes, ov);

// Note that this method leaves pXmitBuf allocated.  It will get deleted in
// FinishDataXfer.
LastError = GetLastError();
return pXmitBuf;
}
```

Example #2 (ISOC endpoints)

```
PUCHAR CCyIsocEndPoint::BeginDirectXfer(PUCHAR buf, LONG bufLen, OVERLAPPED *ov)
{
if ( hDevice == INVALID_HANDLE_VALUE ) return NULL;
int pkts = bufLen / MaxPktSize; // Number of packets implied by bufLen & pktSize
if (bufLen % MaxPktSize) pkts++;
if (pkts == 0) return NULL;
int iXmitBufSize = sizeof (SINGLE_TRANSFER) + (pkts * sizeof(ISO_PACKET_INFO));
UCHAR *pXmitBuf = new UCHAR[iXmitBufSize];
ZeroMemory (pXmitBuf, iXmitBufSize);
PSINGLE_TRANSFER pTransfer = (PSINGLE_TRANSFER) pXmitBuf;
pTransfer->ucEndpointAddress = Address;
pTransfer->IsoPacketOffset = sizeof (SINGLE_TRANSFER);
pTransfer->IsoPacketLength = pkts * sizeof(ISO_PACKET_INFO);
pTransfer->BufferOffset = 0;
pTransfer->BufferLength = 0;
DWORD dwReturnBytes = 0;
DeviceIoControl (hDevice,
      IOCTL_ADAPT_SEND_NON_EP0_DIRECT,
      pXmitBuf, iXmitBufSize,
      buf, bufLen,
      &dwReturnBytes, ov);
// Note that this method leaves pXmitBuf allocated.  It will get deleted in
// FinishDataXfer.
LastError = GetLastError();
return pXmitBuf;
}
```

Example #3 (ISOC endpoints)

```
PUCHAR CCyIsocEndPoint::BeginDirectXfer(PUCHAR buf, LONG bufLen, OVERLAPPED *ov)
{
if ( hDevice == INVALID_HANDLE_VALUE ) return NULL;
int pkts = bufLen / MaxPktSize; // Number of packets implied by bufLen & pktSize
if (bufLen % MaxPktSize) pkts++;
if (pkts == 0) return NULL;
int iXmitBufSize = sizeof (SINGLE_TRANSFER) + (pkts * sizeof(ISO_PACKET_INFO));
UCHAR *pXmitBuf = new UCHAR[iXmitBufSize];
ZeroMemory (pXmitBuf, iXmitBufSize);
PSINGLE_TRANSFER pTransfer = (PSINGLE_TRANSFER) pXmitBuf;
pTransfer->ucEndpointAddress = Address;
pTransfer->IsoPacketOffset = sizeof (SINGLE_TRANSFER);
pTransfer->IsoPacketLength = pkts * sizeof(ISO_PACKET_INFO);
```

```
pTransfer->IsoParams.isoId = USB_ISO_ID;
pTransfer->IsoParams.isoCmd = USB_ISO_CMD_ASAP;
pTransfer->IsoParams.ulParam1 = 0;
DWORD dwReturnBytes = 0;
DeviceIoControl ( hDevice,
            IOCTL_ADAPT_SEND_NON_EP0_DIRECT,
            pXmitBuf, iXmitBufSize,
            buf, bufLen,
            &dwReturnBytes, ov);

// Note that this method leaves pXmitBuf allocated.  It will get deleted in
// FinishDataXfer.
LastError = GetLastError();
return pXmitBuf;
}
```

5.21 IOCTL_ADAPT_SET_DEVICE_POWER_STATE

Description

This IOCTL is no longer supported. It is available to keep backward compatibility with older interface library and application.

Microsoft WDF driver framework manage device power state internally.

5.22 IOCTL_ADAPT_SET_TRANSFER_SIZE

Description

This IOCTL is no longer supported. It is available to keep backward compatibility with older interface library and application.

For more information on USB transfer size please refer link from Microsoft : http://msdn.microsoft.com/en-us/library/ff538112.aspx

Following is the maximum transfer size limit set into the CyUSB3.sys driver for various transfers.

1. Bulk and Interrupt Transfer
   4MBytes
2. Full Speed Isochronous Transfer
   256 Frames
3. High Speed and Super Speed Isochronous Transfer
   1024 Frames

6 CYIOCTL.H

Header
cyioctl.h

Description

A pointer to a SINGLE_TRANSFER structure is passed to the driver for the IOCTL_ADAPT_SEND_NON_EP0_TRANSFER and IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER commands.

The structure is defined as:

```
typedef struct  _SINGLE_TRANSFER {
union {
   SETUP_PACKET    SetupPacket;
   ISO_ADV_PARAMS    IsoParams;
 };
UCHAR Reserved;
UCHAR ucEndpointAddress;
ULONG NtStatus;
ULONG UsbdStatus;
ULONG IsoPacketOffset;
ULONG IsoPacketLength;
ULONG BufferOffset;
ULONG BufferLength;
} SINGLE_TRANSFER, *PSINGLE_TRANSFER;
```

Members

SetupPacket
: Contains required parameters for Control Endpoint transfers,

IsoParams
: Contains optional parameters for Isochronous Endpoint transfers.

reserved
: Reserved. Should be set to 0.

ucEndpointAddress
: Specified the address of the device endpoint in which the transfer will occur.

NtStatus
: NTSTATUS values that are returned by the driver.

UsbdStatus
: USB_STATUS_XXX codes returned from the host controller driver.

IsoPacketOffset
: Specifies the byte offset from the beginning of the structure to an IsoPacket list.

IsoPacketLength
: The length, in bytes, of the IsoPacket list specified at offset IsoPacketOffset.

BufferOffset
: Specifies the byte offset from the beginning of the structure to a transfer buffer.

BufferLength
: The length, in bytes, of the transfer buffer at offset BufferOffset.

6.1 ISO_ADV_PARAMS

Header
cyioctl.h

Description

ISO_ADV_PARAMS is part of the a SINGLE_TRANSFER structure. It contains advanced parameters for Isochronous endpoint transfers when sending the IOCTL_ADAPT_SEND_NON_EP0_TRANSFER and IOCTL_ADAPT_SEND_NON_EP0_DIRECT commands.

The structure is defined as:

```
typedef struct  _ISO_ADV_PARAMS{
                            USHORT isoId;
                            USHORT isoCmd;
                            ULONG ulParam1;
                            ULONG ulParam2;
                            } ISO_ADV_PARAMS, *PISO_ADV_PARAMS;
```

Defines

```
#define USB_ISO_ID                      0x4945
#define USB_ISO_CMD_ASAP                0x8000
#define USB_ISO_CMD_CURRENT_FRAME       0x8001
#define USB_ISO_CMD_SET_FRAME           0x8002
```

Members

isoId
: ISO_ADV_PARAMS structure identifier must be set to USB_ISO_ID.

isoCmd
: Specifies one of the following types of Isoch transfers:

  USB_ISO_CMD_ASAP
  If no transfers have been submitted to the pipe since the pipe was opened or last reset, the transfer to begin on the next frame. Otherwise, the transfer will begin on the first frame following all currently queued requests for the pipe.

  USB_ISO_CMD_CURRENT_FRAME
  Causes the transfer to begin on the current frame number obtained from the host controller driver, plus an optional offset specified in the ulParam1 field.

  USB_ISO_CMD_SET_FRAME
  Causes the transfer to begin on the frame number specified in the ulParam1 field.

ulParam1
: If isoCMD is set to USB_ISO_CMD_ASAP, when the request is returned by the driver this field will contain the frame number that the transfer began on.

  If isoCMD is set to USB_ISO_CMD_CURRENT_FRAME, this field contains the offset from the

current frame number that this transfer will begin on.

If isoCMD is set to USB_ISO_CMD_SET_FRAME, this field contains the frame number that this transfer will begin on.

ulParam2

Reserved. Must be set to 0.

6.2 SINGLE_TRANSFER

Header
cyioctl.h

Description

A pointer to a SINGLE_TRANSFER structure is passed to the driver for the IOCTL_ADAPT_SEND_NON_EP0_TRANSFER and IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER commands.

The structure is defined as:

```
typedef struct  _SINGLE_TRANSFER {
union {
   SETUP_PACKET    SetupPacket;
   ISO_ADV_PARAMS    IsoParams;
 };
UCHAR Reserved;
UCHAR ucEndpointAddress;
ULONG NtStatus;
ULONG UsbdStatus;
ULONG IsoPacketOffset;
ULONG IsoPacketLength;
ULONG BufferOffset;
ULONG BufferLength;
} SINGLE_TRANSFER, *PSINGLE_TRANSFER;
```

Members

SetupPacket
: Contains required parameters for Control Endpoint transfers,

IsoParams
: Contains optional parameters for Isochronous Endpoint transfers.

reserved
: Reserved. Should be set to 0.

ucEndpointAddress
: Specified the address of the device endpoint in which the transfer will occur.

NtStatus
: NTSTATUS values that are returned by the driver.

UsbdStatus
: USB_STATUS_XXX codes returned from the host controller driver.

IsoPacketOffset
: Specifies the byte offset from the beginning of the structure to an IsoPacket list.

IsoPacketLength
: The length, in bytes, of the IsoPacket list specified at offset IsoPacketOffset.

BufferOffset
: Specifies the byte offset from the beginning of the structure to a transfer buffer.

BufferLength
: The length, in bytes, of the transfer buffer at offset BufferOffset.

6.3 SETUP_PACKET

Header
cyioctl.h

Description

A SETUP_PACKET is part of the a SINGLE_TRANSFER structure. It contains important parameters for Control Endpoint transfers when sending the IOCTL_ADAPT_SEND_EP0_TRANSFER command.

The structure is defined as:

```
typedef struct  _SETUP_PACKET {
union  {
 BM_REQ_TYPE  bmReqType;
 UCHAR bmRequest;
};
UCHAR bRequest;
union  {
 WORD_SPLIT  wVal;
 USHORT wValue;
};
union  {
 WORD_SPLIT  wIndx;
 USHORT wIndex;
};
union  {
 WORD_SPLIT  wLen;
 USHORT wLength;
};
ULONG ulTimeOut;
} SETUP_PACKET, *PSETUP_PACKET;
```

6.4 SET_TRANSFER_SIZE_INFO

Header
cyioctl.h

Description

A pointer to a SET_TRANSFER_SIZE_INFO structure is passed to the driver for the IOCTL_ADAPT_GET_TRANSFER_SIZE and IOCTL_ADAPT_SET_TRANSFER_SIZE commands.

The structure is defined as:

```
typedef struct  _SET_TRANSFER_SIZE_INFO {
UCHAR EndpointAddress;
ULONG  TransferSize;
} SET_TRANSFER_SIZE_INFO, *PSET_TRANSFER_SIZE_INFO;
```

7 Features Not Supported

The Following features are not supported by CyUSB3.sys driver due to the lack of interface URBs to the Bus driver.

1. SET ADDRESS Feature

   The SET ADDRESS Request cannot be implemented through control endpoint.

2. SYNC FRAME

   The SYNC FRAME Request cannot be implemented through Control Endpoint.

3. USB3.0 Bulk streaming.

4. Following IOCTLs are not supported.

   To get more detail on each IOCTL please refer below link.

Get Device PowerState
Set Device PowerState
Set Transfer size
Get Transfer size

8 EZ-USB.sys to CyUSB3.sys Migration Guide

This chapter provides comparison between ezusb.sys and cyusb3.sys driver IOCTL interface. If you are using the ezusb.sys and wants migrate to cyusb3.sys to take advantage of advance feature supported by cyusb3.sys, please refer below table. This table shows one-o-one mapping of two drivers IOCTL interface.

IOCTL Mapping table

ezusb.sys IOCTL interface	CyUSB3.sys IOCTL interface	Description
IOCTL_Ezusb_RESETPIPE	IOCTL_ADAPT_RESET_PIPE	Direct equivalent
IOCTL_Ezusb_ABORTPIPE	IOCTL_ADAPT_ABORT_PIPE	
IOCTL_Ezusb_SETINTERFACE	IOCTL_ADAPT_SELECT_INTERFACE	
IOCTL_EZUSB_GET_DRIVER_ VERSION	IOCTL_ADAPT_GET_DRIVER_VERSION	
IOCTL_EZUSB_GET_CURRENT_FRAME_NUMBER	IOCTL_ADAPT_GET_CURRENT_FRAME	
IOCTL_Ezusb_RESET	IOCTL_ADAPT_RESET_PARENT_PORT	
IOCTL_EZUSB_BULK_READ	IOCTL_ADAPT_SEND_NON_EP0_TRANSFER / IOCTL_ADAPT_SEND_NON_EP0_DIRECT	Indirect equivalent. CyUSB3.sys has two IOCTLs to implement non-control transfer rather than the ezusb.sys method of having one IOCTL per transfer type and direction. So all non-control transfer based IOCTL of ezusb.sys map to two IOCTLs of CyUSB3.sys
IOCTL_EZUSB_BULK_WRITE		
IOCTL_EZUSB_ISO_READ		
IOCTL_EZUSB_ISO_WRITE		
IOCTL_Ezusb_GET_DEVICE_DESCRIPTOR	IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER	Indirect equivalent. CyUSB3.sys has single IOCTL to implement control transfer rather than the ezusb.sys method of having one IOCTL per type and one IOCTL per standard request. All control transfer based IOCTL of ezusb.sys map to single IOCTL of CyUSB3.sys
IOCTL_Ezusb_GET_CONFIGURATION_DESCRIPTOR		
IOCTL_Ezusb_VENDOR_REQU EST		
IOCTL_Ezusb_GET_STRING_DESCRIPTOR		

IOCTL_Ezusb_ANCHOR_DOWNLOAD		
IOCTL_EZUSB_ANCHOR_DOWNLOAD		
IOCTL_EZUSB_VENDOR_OR_CLASS_REQUEST		
IOCTL_EZUSB_SET_FEATURE		
IOCTL_Ezusb_GET_PIPE_INFO	IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER, IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING	Indirect equivalent. Parse the interface descriptor of the device to populate the output buffer with data of the PUSB_INTERFACE_INFORMATION structure that will be returned by ezusb.sys
IOCTL_EZUSB_START_ISO_STREAM	IOCTL_ADAPT_SEND_NON_EP0_TRANSFER / IOCTL_ADAPT_SEND_NON_EP0_DIRECT	No direct equivalent ezusb.sys implements streaming on isochronous endpoint at the driver level. While CyUSB3.sys provide single IOCTL to send/ received Isochronous packets.
IOCTL_EZUSB_STOP_ISO_STREAM		
IOCTL_EZUSB_READ_ISO_BUFFER		
IOCTL_EZUSB_GET_LAST_ERROR	N/A	Windows Win32 API GetLastError() API provides the error code for last transfer.

Get driver handle in application using CyUSB3.sys

Modify CyUSB3.INF file

CyUSB3.sys IOCTLs definition is provided in the cyioctl.h header file.

Index

- • -

- C -

- S -

- W -