

Cypress CyUsb.sys Programmer's Reference

© 2003 Cypress Semiconductor

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Table of Contents

Part I Driver Overview	3
Part II Modifying CyUSB.INF	3
Part III Matching Devices to the Driver	7
1 Windows 2000	7
2 Windows XP	9
Part IV The IOCTL Interface	12
1 Getting a Handle to the Driver	13
2 IOCTL_ADAPT_ABORT_PIPE	14
3 IOCTL_ADAPT_CYCLE_PORT	15
4 IOCTL_ADAPT_GET_ADDRESS	15
5 IOCTL_ADAPT_GET_ALT_INTERFACE_SETTING	15
6 IOCTL_ADAPT_GET_DEVICE_NAME	16
7 IOCTL_ADAPT_GET_DEVICE_POWER_STATE	16
8 IOCTL_ADAPT_GET_DRIVER_VERSION	17
9 IOCTL_ADAPT_GET_FRIENDLY_NAME	17
10 IOCTL_ADAPT_GET_NUMBER_ENDPOINTS	18
11 IOCTL_ADAPT_GET_TRANSFER_SIZE	18
12 IOCTL_ADAPT_GET_USBDI_VERSION	18
13 IOCTL_ADAPT_RESET_PARENT_PORT	19
14 IOCTL_ADAPT_RESET_PIPE	19
15 IOCTL_ADAPT_SELECT_INTERFACE	20
16 IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER	20
17 IOCTL_ADAPT_SEND_NON_EP0_TRANSFER	21
18 IOCTL_ADAPT_SEND_NON_EP0_DIRECT	22
19 IOCTL_ADAPT_SET_DEVICE_POWER_STATE	24
20 IOCTL_ADAPT_SET_TRANSFER_SIZE	24
Part V CYIOCTL.H	25
1 SINGLE_TRANSFER	25
2 SETUP_PACKET	26
3 SET_TRANSFER_SIZE_INFO	26

1 Driver Overview

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

CYUSB.SYS is a USB device driver for Windows 2000 and Windows XP that is capable of communicating with any USB 2.0 compliant device. 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.

However, the driver would be ideal for communicating with a vendor-specific device from a custom USB application. Or, it might be used 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, provides a high-level programming interface to the driver. This help file documents the low-level, more cumbersome and explicit, programming interface.

Features

- Windows Driver Model (WDM) compliant
- WHQL Certified (not signed)
- Compatible with any USB 2.0 compliant device
- Supports Windows PnP and Power Management level S4
- Supports USB Remote Wake-up
- Supports Control, Bulk, Interrupt and Isochronous endpoints
- 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 uframe
- Supports automatic [play-back of control transfer scripts](#) at device startup

2 Modifying CyUSB.INF

The CYUSB.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](#)
4. [Execute a saved script](#) of commands at driver load time

Add a device's identifiers to the driver

To make the driver match to a specific device, the device's vendor ID and product ID need to be added to the .inf file.

Locate the **[Cypress]** section and duplicate the line
;%VID_VVVV&PID_PPPP.DeviceDesc%=CyUSB,USB\VID_VVVV&PID_PPPP

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the **VendorID** for the device

Change the **PPPP** 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 [Cypress] section like the following

```
%VID_04B4&PID_DE01.DeviceDesc%=CyUSB, USB\VID_04B4&PID_DE01
```

Now, move to the bottom of the CYUSB.INF file and locate the [Strings] section and duplicate the line.

```
;VID_VVVV&PID_PPPP.DeviceDesc="My Device Description"
```

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the **VendorID** for the device

Change the **PPPP** 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 OTG DE1 DevBoard"
```

Replace Cypress strings

If you plan to do more than just add your device's VID/PID to the CYUSB.INF file, it is strongly recommended that you create your own .INF file and a copy of CYUSB.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 CYUSB.SYS.

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

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

Change the quoted **PROVIDER** string.

Change the quoted **MFGNAME** string.

Change the quoted **CyUSB.SvcDesc** string.

There is also an identifier named Cypress that is used in the file. This can be modified if you want no mention of Cypress in the .inf file.

Locate the **[Manufacturer]** section near the top of the file.

Change the symbolic name, **Cypress**, in the line

```
%MFGNAME%=Cypress
```

Change the section name, **[Cypress]** in the next line to match the **symbolic name** used in the **%MFGNAME%=** assignment.

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 CYUSB.SYS from different hardware vendors can exist on a given system without colliding.

To change the driver's GUID,

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

Locate the **[Strings]** section in the CyUSB.inf file

Locate the line

CyUSB.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 CYUSB.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

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

Save the script as a file named CYUSB.SPT

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

Locate and uncomment the 6 lines in the .inf file that are preceeded by the comment

-----Uncomment below to support script file processing-----

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.

The following is an excerpt from a .inf file that plays a script called MyDevice.spt when VID/PID of 04B4/8613 is enumerated. If VID/PID 0547/1002 enumerates, the script is not played. (This .inf is compatible with WinXP and Win2k.)

Note the separate blocks of declarations for MYDEVICE and CYUSB.

```

[Cypress]
%VID_04B4&PID_8613.DeviceDesc%=MYDEVICE, USB\VID_04B4&PID_8613
%VID_0547&PID_1002.DeviceDesc%=CYUSB, USB\VID_0547&PID_1002

[DestinationDirs]
CYUSB.Files = 10,System32\Drivers
MYDEVICE.Files = 10,System32\MYDEVICE

;=====

[MYDEVICE.Files]
MYDEVICE.SPT

[MYDEVICE.NT]
CopyFiles=CYUSB.Files, MYDEVICE.Files
AddReg=CYUSB.AddReg

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

[MYDEVICE.NT.Services]
Addservice = CYUSB, 0x00000002, CYUSB.AddService

[MYDEVICE.AddReg.Guid]
HKR,,DriverGUID,,%CYUSB.GUID%
HKR,,DriverEXECSRIPT,,%MYDEVICE.EXECSRIPT%

;=====

[CYUSB.Files]
CYUSB.SYS

[CYUSB.NT]
CopyFiles=CYUSB.Files
AddReg=CYUSB.AddReg

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

[CYUSB.NT.Services]
Addservice = CYUSB, 0x00000002, CYUSB.AddService

[CYUSB.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,CyUsb.sys

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

;=====

[CYUSB.AddService]
DisplayName = %CYUSB.SvcDesc%
ServiceType = 1 ; SERVICE_KERNEL_DRIVER
StartType = 3 ; SERVICE_DEMAND_START
ErrorControl = 1 ; SERVICE_ERROR_NORMAL
ServiceBinary = %10%\System32\Drivers\CYUSB.SYS
LoadOrderGroup = Base

[Strings]
PROVIDER="Cypress"
MFGNAME="Cypress Semiconductor"
CYUSB_INSTALL="Cypress Installation Disk"
VID_04B4&PID_8613.DeviceDesc="My first device"
VID_0547&PID_1002.DeviceDesc="My re-programmed Device"

```

```
CYUSB.SvcDesc="Cypress USB Device"
CYUSB.GUID="{AE18AA60-7F6A-11d4-97DD-00010229B959}"
MYDEVICE.EXECSCRIPT="\systemroot\system32\MYDEVICE\MYDEVICE.spt"
```

3 Matching Devices to the Driver

Usually, matching of a USB device to the CYUSB.SYS driver will need to be manually configured. This configuration consist of two steps.

- Step 1 :** Add the device's VendorID and ProductID to the CYUSB.INF file.
Step 2 : Force WindowsXP to use the CYUSB.SYS driver with the device.

Though similar, these steps are slightly different for [Windows 2000](#) and [WinXP](#).

3.1 Windows 2000

Usually, matching of a USB device to the CYUSB.SYS driver will need to be manually configured. This configuration consist of two steps.

- Step 1 :** Add the device's VendorID and ProductID to the CYUSB.INF file.

After installation of the Developer Studio files, the driver is located in a Driver subdirectory of the install directory. (Default is C:\Program Files\Cypress\DevStudio\Driver.)

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

Locate the **[Cypress]** section and duplicate the line
 ;%VID_ VVVV &PID_ PPPP .DeviceDesc%=CyUSB, USB\VID_ VVVV &PID_ PPPP

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the VendorID for the device

Change the **PPPP** 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 [Cypress] section like the following:

```
%VID_04B4&PID_DE01.DeviceDesc%=CyUSB, USB\VID_04B4&PID_DE01
```

Now, move to the bottom of the CyUSB.inf file and locate the **[Strings]** section and duplicate the line.

```
; VID_ VVVV &PID_ PPPP .DeviceDesc="My Device Description "
```

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the VendorID for the device

Change the **PPPP** 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 OTG DE1 DevBoard"

Save the file.

Step 2 : Force Windows2000 to use the CYUSB.SYS driver with the device.

Connect the device to the PC

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

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 CYUSB.SYS driver was selected and, if not, manually instruct Windows to use that driver.

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

In the **Computer Management** window, select **Device Manager**

In the right window pane, click the + icon next to **Universal Serial Bus controllers**

Locate your device in the list and double click on it

Select the **Driver** tab in the Properties dialog that comes up

Click on the **Driver Details** button.

*If the displayed driver file is CYUSB.SYS, Windows has already matched the device to this driver and you should click **OK** and **Cancel** . If not, proceed with the remaining steps.*

Click **OK**

Select **Update Driver**

Click **Next**

Select **Search for a suitable driver for my device (recommended)**

Click **Next**

Select **Specify a location**

Click **Next**

Navigate to the directory containing CYUSB.SYS

CYUSB.INF should be automatically placed in the File name field

Click **Open**

Click **OK**

Click **Next**

Click **Finish**

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 XP

Usually, matching of a USB device to the CYUSB.SYS driver will need to be manually configured. This configuration consist of two steps.

Step 1 : Add the device's VendorID and ProductID to the CYUSB.INF file.

After installation of the Developer Studio files, the driver is located in a Driver subdirectory of the install directory. (Default is C:\Program Files\Cypress\DevStudio\Driver.)

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

Locate the **[Cypress]** section and duplicate the line
; %VID_ **VVVV** &PID_ **PPPP** .DeviceDesc%=CyUSB, USB\VID_ **VVVV** &PID_ **PPPP**

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the VendorID for the device

Change the **PPPP** 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 [Cypress] section like the following

```
%VID_04B4&PID_DE01.DeviceDesc%=CyUSB, USB\VID_04B4&PID_DE01
```

Now, move to the bottom of the CyUSB.inf file and locate the **[Strings]** section and duplicate the line.

```
; VID_ VVVV &PID_ PPPP .DeviceDesc="My Device Description "
```

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the VendorID for the device

Change the **PPPP** 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 OTG DE1 DevBoard"
```

Save the file.

Step 2 : Force WindowsXP to use the CYUSB.SYS driver with the device.

Connect the device to the PC

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

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 CYUSB.SYS driver was selected and, if not, manually instruct Windows to use that driver.

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

In the **Computer Management** window, select **Device Manager**

In the right window pane, click the + icon next to **Universal Serial Bus controllers**

Locate your device in the list and double click on it

Select the **Driver** tab in the Properties dialog that comes up

Click on the **Driver Details** button.

*If the displayed driver file is CYUSB.SYS, Windows has already matched the device to this driver and you should click **OK** and **Cancel** . If not, proceed with the remaining steps.*

Click **OK**

Click **Update Driver**

Select **Install from a list or specific location (Advanced)**

Click **Next**

Select **Don't search. I will choose the driver to install.**

Click **Next**

Click **Have Disk**

Click **Browse**

Navigate to the directory containing CYUSB.SYS

CYUSB.INF should be automatically placed in the File name field

Click **Open**

Click **OK**

Click **Next**

Click **Finish**

Click **Close**

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

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

This configuration consist of two steps.

Step 1 : Add the device's VendorID and ProductID to the CYUSB.INF file.

After installation of the SuiteUSB files, the driver is located in a Driver subdirectory of the install directory. (Default is C:\Program Files\Cypress\SuiteUSB\Driver.)

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

Locate the **[Cypress]** section and duplicate the line
; %VID_ VVVV &PID_ PPPP .DeviceDesc%=CyUSB, USB\VID_ VVVV &PID_ PPPP

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the VendorID for the device

Change the **PPPP** 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 [Cypress] section like the following

```
%VID_04B4&PID_DE01.DeviceDesc%=CyUSB, USB\VID_04B4&PID_DE01
```

Now, move to the bottom of the CyUSB.inf file and locate the **[Strings]** section and duplicate the line.

```
; VID_ VVVV &PID_ PPPP .DeviceDesc="My Device Description "
```

Remove the semicolon from the duplicate line

Change the **VVVV** to contain the hexadecimal value of the VendorID for the device

Change the **PPPP** 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 OTG DE1 DevBoard"
```

Save the file.

Step 2 : Force WindowsXP to use the CYUSB.SYS driver with the device.

Connect the device to the PC

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

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 CYUSB.SYS driver was selected and, if not, manually instruct Windows to use that driver.

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

In the **Computer Management** window, select **Device Manager**

In the right window pane, click the + icon next to **Universal Serial Bus controllers**

Locate your device in the list and double click on it

Select the **Driver** tab in the Properties dialog that comes up

Click on the **Driver Details** button.

*If the displayed driver file is CYUSB.SYS, Windows has already matched the device to this driver and you should click **OK** and **Cancel** . If not, proceed with the remaining steps.*

Click **OK**

Click **Update Driver**

Select **Install from a list or specific location (Advanced)**

Click **Next**

Select **Don't search. I will choose the driver to install.**

Click **Next**

Click **Have Disk**

Click **Browse**

Navigate to the directory containing CYUSB.SYS

CYUSB.INF should be automatically placed in the File name field

Click **Open**

Click **OK**

Click **Next**

Click **Finish**

Click **Close**

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

4 The IOCTL Interface

Applications software communicates with the CYUSB.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 CYUSB.SYS driver.

Example

```

DWORD dwBytes = 0;
UCHAR EndptAddress = 0x82;

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

```

4.1 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_INTERFACEDEVIC
E);

```

```

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);
        }
    }
}

```

4.2 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);
```

4.3 IOCTL_ADAPT_CYCLE_PORT

Description

This command causes the USB device to be logically disconnected from the bus and, then, re-connected.

NULL pointers are passed to DeviceIoControl in the lpInBuffer and lpOutBuffer parameters.

Example

```
DWORD dwBytes = 0;

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

4.4 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);
```

4.5 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 intfci = 0;
UCHAR alt;

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

4.6 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;
```

4.7 IOCTL_ADAPT_GET_DEVICE_POWER_STATE

Description

This command retrieves the power state of the device

A pointer to a `ULONG` variable (`pwrState`) is passed as both the `lpInBuffer` and `lpOutBuffer` parameters to the `DeviceIoControl()` function.

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

Possible return values for the `pwrState` are:

- 1 => Power State D0 (Full On)
- 2 => Power State D1
- 3 => Power State D2
- 4 => Power State D3 (Full Asleep)

Example

```

DWORD dwBytes = 0;
ULONG pwrState;

DeviceIoControl(hDevice, IOCTL_ADAPT_GET_DEVICE_POWER_STATE,
               &pwrState, sizeof (pwrState),
               &pwrState, sizeof (pwrState),
               &dwBytes, NULL);

```

4.8 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);

```

4.9 IOCTL_ADAPT_GET_FRIENDLY_NAME**Description**

This command retrieves the string associated with the device in the [Strings] section of the CyUSB.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;
PCHAR FriendlyName = new UCHAR[256];

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

delete[] FriendlyName;

```

4.10 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);
```

4.11 IOCTL_ADAPT_GET_TRANSFER_SIZE

Description

This command retrieves the current transfer size for a given endpoint. The transfer size is not the same as the `MaxPacketSize` for the endpoint. Rather, the transfer size is always an integral multiple of the endpoint's `MaxPacketSize`.

A pointer to a `SET_TRANSFER_SIZE_INFO` structure is passed as both the `lpInBuffer` and `lpOutBuffer` parameters to the `DeviceIoControl()` function. This structure must be pre-loaded with the address of the endpoint of interest. Upon return, the structure will contain the transfer size of endpoint.

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

Example

```
DWORD BytesXfered;
SET_TRANSFER_SIZE_INFO SetTransferInfo;
SetTransferInfo.EndpointAddress = Address;

DeviceIoControl(hDevice, IOCTL_ADAPT_GET_TRANSFER_SIZE,
                &SetTransferInfo, sizeof (SET_TRANSFER_SIZE_INFO),
                &SetTransferInfo, sizeof (SET_TRANSFER_SIZE_INFO),
                &BytesXfered, NULL);

LONG transferSz = SetTransferInfo.TransferSize;
```

4.12 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);
```

4.13 IOCTL_ADAPT_RESET_PARENT_PORT

Description

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

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);
```

4.14 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);
```

4.15 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);
```

4.16 IOCTL_ADAPT_SEND_EP0_CONTROL_TRANSFER

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.

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->WaitForever = false;
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);

```

4.17 IOCTL_ADAPT_SEND_NON_EP0_TRANSFER

Description

NOTE:

With the release of CyUSB.sys version 1.5.503.0, the faster `IOCTL_ADAPT_SEND_NON_EP0_DIRECT` should be used instead of this command. `IOCTL_ADAPT_SEND_NON_EP0_TRANSFER` remains supported only to provide driver compatibility to existing applications that use it.

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.

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->WaitForever = false;
    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;
}

```

4.18 IOCTL_ADAPT_SEND_NON_EP0_DIRECT

Description

This IOCTL is only exposed by the CyUSB.sys driver version 1.5.503.0 or later. It optimizes throughput by using separate buffers for the SINGLE_TRANSFER structure and the transfer data. (The CyAPI.lib class library, version 1.0.5.0, uses this command, instead of the slower IOCTL_ADAPT_SEND_NON_EP0_TRANSFER, if it detects a capable driver.)

This command 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

For ISOC transfers using this command, the following constraints apply:

- 1) The endpoint transfer size must be a multiple of the endpoint's MaxPacketSize * 8. (See IOCTL_ADAPT_SET_TRANSFER_SIZE.)
- 2) The buffer length parameter (bufLen in the below examples) must also be a multiple of the endpoint's MaxPacketSize * 8.
- 3) The SINGLE_TRANSFER structure must be followed by additional space sufficient to hold the PACKET_INFO structures for the transfer (see example #2, 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->WaitForever = false;
}

```

```

pTransfer->ucEndpointAddress = Address;
pTransfer->IsoPacketLength = 0;
pTransfer->BufferOffset = 0;
pTransfer->BufferLength = 0;

DWORD dwReturnBytes;
DeviceIoControl (hDevice,
                 IOCTL_ADAPT_SEND_NON_EPO_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->WaitForever = false;
    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_EPO_DIRECT,
                    pXmitBuf, iXmitBufSize,
                    buf, bufLen,
                    &dwReturnBytes, ov);

    // Note that this method leaves pXmitBuf allocated. It will get deleted in
    // FinishDataXfer.

    LastError = GetLastError();
    return pXmitBuf;
}

```


4.19 IOCTL_ADAPT_SET_DEVICE_POWER_STATE

Description

This command sets the power state of the device.

A pointer to a ULONG variable (pwrState) is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl() function.

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

Valid values for the pwrState are:

- 1 => Power State D0 (Full On)
- 2 => Power State D1
- 3 => Power State D2
- 4 => Power State D3 (Full Asleep)

Example

```
// Put the device into full asleep (Device Power State D3)
DWORD dwBytes = 0;
ULONG pwrState = 4;

DeviceIoControl(hDevice, IOCTL_ADAPT_SET_DEVICE_POWER_STATE,
                &pwrState, sizeof (pwrState),
                &pwrState, sizeof (pwrState),
                &dwBytes, NULL);
```

4.20 IOCTL_ADAPT_SET_TRANSFER_SIZE

Description

This command sets the transfer size for a given endpoint. The transfer size is not the same as the MaxPacketSize for the endpoint. Rather, the transfer size is always an integral multiple of the endpoint's MaxPacketSize.

Small transfer sizes are memory efficient but result in multiple operations to effect a data transfer. Larger transfer sizes are more wasteful of memory, but accomplish larger data transfers with fewer IO transactions.

A pointer to a SET_TRANSFER_SIZE_INFO structure is passed as both the lpInBuffer and lpOutBuffer parameters to the DeviceIoControl() function. This structure contains the address of the endpoint that is to be changed and the new transfer size.

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

Example

```
DWORD BytesXfered;
SET_TRANSFER_SIZE_INFO SetTransferInfo;
SetTransferInfo.EndpointAddress = Address;
SetTransferInfo.TransferSize = 0x2000; // An 8 KB transfer size

DeviceIoControl(hDevice, IOCTL_ADAPT_SET_TRANSFER_SIZE,
                &SetTransferInfo, sizeof (SET_TRANSFER_SIZE_INFO),
```

```
&SetTransferInfo, sizeof (SET_TRANSFER_SIZE_INFO),
&BytesXfered, NULL);
```

5 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 {
    SETUP_PACKET SetupPacket;
    IN BOOLEAN WaitForever;
    UCHAR ucEndpointAddress;
    ULONG NtStatus;
    ULONG UsbdStatus;
    ULONG IsoPacketOffset;
    ULONG IsoPacketLength;
    ULONG BufferOffset;
    ULONG BufferLength;
} SINGLE_TRANSFER, *PSINGLE_TRANSFER;
```

5.1 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 {
    SETUP_PACKET SetupPacket;
    IN BOOLEAN WaitForever;
    UCHAR ucEndpointAddress;
    ULONG NtStatus;
    ULONG UsbdStatus;
    ULONG IsoPacketOffset;
    ULONG IsoPacketLength;
    ULONG BufferOffset;
    ULONG BufferLength;
} SINGLE_TRANSFER, *PSINGLE_TRANSFER;
```

5.2 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_EPO_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;
```

5.3 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;
```

