

# **Cypress CyUSB .NET Programmer's Reference**

© 2018 Cypress Semiconductor

# Table of Contents

<b>Part I Overview</b>	<b>7</b>
<b>Part II CyUSB Library Class Hierarchy</b>	<b>9</b>
<b>Part III New Features</b>	<b>10</b>
1 USB3.0 Support Overview.....	11
<b>Part IV CyUSB</b>	<b>12</b>
1 CyBOS_CONTAINER_ID.....	13
Tree .....	14
Length .....	15
DescriptorType .....	16
DevCapabilityType .....	17
Reserved .....	18
ContainerID .....	19
ToString() .....	20
2 CyBOS_SS_DEVICE_CAPABILITY.....	21
Tree .....	22
Length .....	23
DescriptorType .....	24
DevCapabilityType .....	25
bmAttribute .....	26
SpeedsSupported .....	27
FunctionalitySupport .....	28
U1DevExitLat .....	29
U2DevExitLat .....	30
ToString() .....	31
3 CyBOS_USB20_DEVICE_EXT.....	32
Tree .....	33
Length .....	34
DescriptorType .....	35
DevCapabilityType .....	36
bmAttribute .....	37
ToString() .....	38
4 CyUSBBOS.....	39
USB20_DeviceExt .....	41
SS_DeviceCap .....	42
Container_ID .....	43
Tree .....	44
Length .....	45
DescriptorType .....	46
TotalLength .....	47
NumDeviceCaps .....	48
ToString() .....	49

<b>5</b>	<b>CyBulkEndPoint.....</b>	<b>50</b>
<b>6</b>	<b>CyConst.....</b>	<b>51</b>
	DEVICES_CYUSB.....	52
	DEVICES_HID .....	53
	DEVICES_MSC .....	54
	DIR_FROM_DEVICE .....	55
	DIR_TO_DEVICE .....	56
	REQ_CLASS .....	57
	REQ_STD .....	58
	REQ_VENDOR .....	59
	SINGLE_XFER_LEN.....	60
	TGT_DEVICE .....	61
	TGT_ENDPT .....	62
	TGT_INTFC .....	63
	TGT_OTHER .....	64
	INFINITE .....	65
	FX3_FWDWNLOAD_MEDIA_TYPE.....	66
	FX3_FWDWNLOAD_ERROR_CODE.....	67
<b>7</b>	<b>CyControlEndPoint.....</b>	<b>68</b>
	Read( ) .....	69
	Write( ) .....	70
	XferData( ) .....	71
	Direction .....	72
	Index .....	73
	ReqCode .....	74
	ReqType .....	75
	Target .....	76
	Value .....	77
<b>8</b>	<b>CyFX2Device.....</b>	<b>78</b>
	LoadEEPROM .....	79
	LoadRAM .....	81
	Reset .....	83
<b>9</b>	<b>CyFX3Device.....</b>	<b>84</b>
	DownloadFw( ) .....	85
	IsBootLoaderRunning( ) .....	86
	GetFwErrorString( ) .....	87
<b>10</b>	<b>CyInterruptEndPoint.....</b>	<b>88</b>
<b>11</b>	<b>CyIsocEndPoint.....</b>	<b>89</b>
	BeginDataXfer( ) .....	90
	FinishDataXfer( ) .....	92
	GetPktBlockSize( ) .....	94
	GetPktCount( ) .....	96
	XferData( ) .....	97
	XferData( ) .....	98
<b>12</b>	<b>CyUSBConfig.....</b>	<b>99</b>
	ToString( ) .....	103
	AltInterfaces .....	107
	bConfigurationValue .....	108
	bDescriptorType .....	109
	bLength .....	110
	bmAttributes .....	111

bNumInterfaces .....	112
iConfiguration .....	113
MaxPower .....	114
Tree .....	115
wTotalLength .....	116
Interfaces .....	117
<b>13 CyUSBDevice .....</b>	<b>121</b>
EndPointOf( ) .....	122
GetConfigDescriptor( ) .....	123
GetDeviceDescriptor( ) .....	124
GetIntfcDescriptor( ) .....	125
GetBosDescriptor() .....	126
GetBosUSB20DeviceExtensionDescriptor() .....	127
GetBosSSCapabilityDescriptor() .....	128
GetBosContainerIDDescriptor() .....	129
ReConnect( ) .....	130
Reset( ) .....	131
ToString( ) .....	132
UsbdStatusString( ) .....	136
AltIntfc .....	137
AltIntfcCount .....	138
bHighSpeed .....	142
bSuperSpeed .....	143
BcdDevice .....	144
Config .....	145
ConfigAttrib .....	146
ConfigCount .....	147
ConfigValue .....	148
DeviceHandle .....	149
DriverVersion .....	150
EndPointCount .....	151
IntfcClass .....	154
IntfcProtocol .....	155
IntfcSubClass .....	156
MaxPacketSize .....	157
MaxPower .....	158
StrLangID .....	159
Tree .....	160
USBIDVersion .....	161
BulkInEndPt .....	162
BulkOutEndPt .....	163
ControlEndPt .....	164
EndPoints .....	165
InterruptInEndPt .....	168
InterruptOutEndPt .....	169
IsocInEndPt .....	170
IsocOutEndPt .....	171
USBCfgs .....	172
<b>14 CyUSBEndPoint .....</b>	<b>173</b>
Abort( ) .....	174
BeginDataXfer( ) .....	175
FinishDataXfer( ) .....	178
Reset( ) .....	181

ToString( ) .....	182
WaitForXfer( ) .....	185
XferData( ) .....	188
Address .....	189
Attributes .....	190
bln .....	191
BytesWritten .....	192
DscLen .....	193
DscType .....	194
hDevice .....	195
Interval .....	196
MaxPktSize .....	197
NtStatus .....	198
TimeOut .....	199
Tree .....	200
UsbdStatus .....	201
XferMode .....	202
XferSize .....	203
SSDscLen .....	204
SSDscType .....	205
SSMaxBurst .....	206
SSBmAttribute .....	207
SSBytePerInterval .....	208
<b>15 CyUSBInterface.....</b>	<b>209</b>
ToString .....	213
bAlternateSetting .....	217
bAltSettings .....	218
bDescriptorType .....	219
bInterfaceClass .....	220
bInterfaceNumber .....	221
bInterfaceProtocol .....	222
bInterfaceSubClass .....	223
bLength .....	224
bNumEndpoints .....	225
iInterface .....	226
Tree .....	227
wTotalLength .....	228
EndPoints .....	229
<b>16 CyUSBStorDevice.....</b>	<b>232</b>
SendScsiCmd( ) .....	233
ToString( ) .....	234
BlockSize .....	235
TimeOut .....	236
<b>17 ISO_PKT_INFO.....</b>	<b>237</b>
<b>18 OVERLAPPED.....</b>	<b>238</b>
<b>19 OverlapSignalAllocSize.....</b>	<b>239</b>
<b>20 PInvoke.....</b>	<b>240</b>
CreateEvent( ) .....	241
WaitForSingleObject( ) .....	242
<b>21 USB_CONFIGURATION_DESCRIPTOR.....</b>	<b>243</b>
<b>22 USB_DEVICE_DESCRIPTOR.....</b>	<b>244</b>

23	USB_INTERFACE_DESCRIPTOR.....	245
24	USB_BOS_USB20_DEVICE_EXTENSION.....	246
25	USB_BOS_SS_DEVICE_CAPABILITY.....	247
26	USB_BOS_CONTAINER_ID.....	248
27	USB_BOS_DESCRIPTOR.....	249
28	USBDevice .....	250
	Dispose( ) .....	251
	Equals( ) .....	252
	BcdUSB .....	253
	DevClass .....	254
	DevProtocol .....	255
	DevSubClass .....	256
	DriverName .....	257
	FriendlyName .....	258
	Manufacturer .....	259
	Name .....	260
	Path .....	261
	Product .....	262
	ProductID .....	263
	SerialNumber .....	264
	Tree .....	265
	USBAddress .....	266
	VendorID .....	267
29	USBDeviceList.....	268
	DeviceAttached( ) .....	270
	DeviceRemoved() .....	271
	Dispose( ) .....	272
	USBDeviceList( ) .....	273
	Count .....	274
	USBDeviceList [int index] .....	275
	USBDeviceList [string fName] .....	276
	USBDeviceList [int VID, int PID] .....	277
	USBDeviceList [string sMfg, string sProd] .....	278
30	Util .....	279
	ParseHexData( ) .....	280
	ParseHexFile( ) .....	282
	ParseIICData( ) .....	284
	ParseIICFile( ) .....	286
	ReverseBytes( ) .....	288
	ReverseBytes( ) .....	289
	Assemblies .....	290
	MaxFwSize .....	291
31	XMODE.....	292

## Part V Features Not Supported

293

## Index

294

# 1 Overview

## Library Overview

[Top](#) [Next](#)

*CyUSB.dll* is a managed Microsoft .NET class library. It provides a high-level, powerful programming interface to USB devices.

Rather than communicate with USB device drivers directly via Win32 API calls such as *SetupDiXxxx* and *DeviceIoControl*, applications can access USB devices via library methods such as [XferData](#) and properties such as [AltIntfc](#).

Because *CyUSB.dll* is a managed .NET library, its classes and methods can be accessed from any of the Microsoft Visual Studio.NET managed languages such as Visual Basic.NET, C#, Visual J# and managed C++.

To use the library, you need to add a reference to *CyUSB.dll* to your project's References folder. Then, any source file that accesses the CyUSB namespace will need to include a line to include the namespace in the appropriate syntax.

Examples:

### **Visual Basic.net**

```
Imports CyUSB
```

### **Visual C#**

```
using CyUSB;
```

### **Visual C++ (Win Forms App)**

```
using namespace CyUSB;
```

### **Visual J#**

```
import CyUSB.*;
```

The library employs a model of *DeviceList*, *Devices* and *EndPoints*. An application will normally create an instance of the [USBDeviceList](#) class which represents a list of USB devices. Each of those devices can then be accessed individually.

Commonly, the devices represented in the device list will be vendor-specific USB devices (i.e. non USB Class devices) served by the *CyUSB3.sys* driver. Such members of the device list will be instances of the [CyUSBDevice](#) class and will expose one or more [CyUSBEndPoints](#) through which data transfers can be performed.

It is also possible to populate a *USBDeviceList* with objects representing USB class devices.

Once a *USBDeviceList* object has been successfully instantiated, specific devices in the list can be quickly accessed using one of several "[indexers](#)" into the list. This model makes locating and accessing USB devices very straight-forward.

Windows PlugNPlay (PnP) events are also easily supported by the library.

The C# code example below demonstrates creation of a *USBDeviceList*, setting-up the handling of PnP events and location of a specific device in the device list.

**C# Example:**

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
using CyUSB;

public partial class Form1 : Form
{
    USBDeviceList usbDevices;
    CyUSBDevice myDevice;

    public Form1()
    {
        InitializeComponent();

        usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
        usbDevices.DeviceAttached += new EventHandler(usbDevices_DeviceAttached);
        usbDevices.DeviceRemoved += new EventHandler(usbDevices_DeviceRemoved);

        // Get the first device having VendorID == 0x04B4 and ProductID == 0x8613
        myDevice = usbDevices[0x04B4, 0x8613] as CyUSBDevice;
        if (myDevice != null)
            StatusLabel.Text = myDevice.FriendlyName + " connected.";
    }

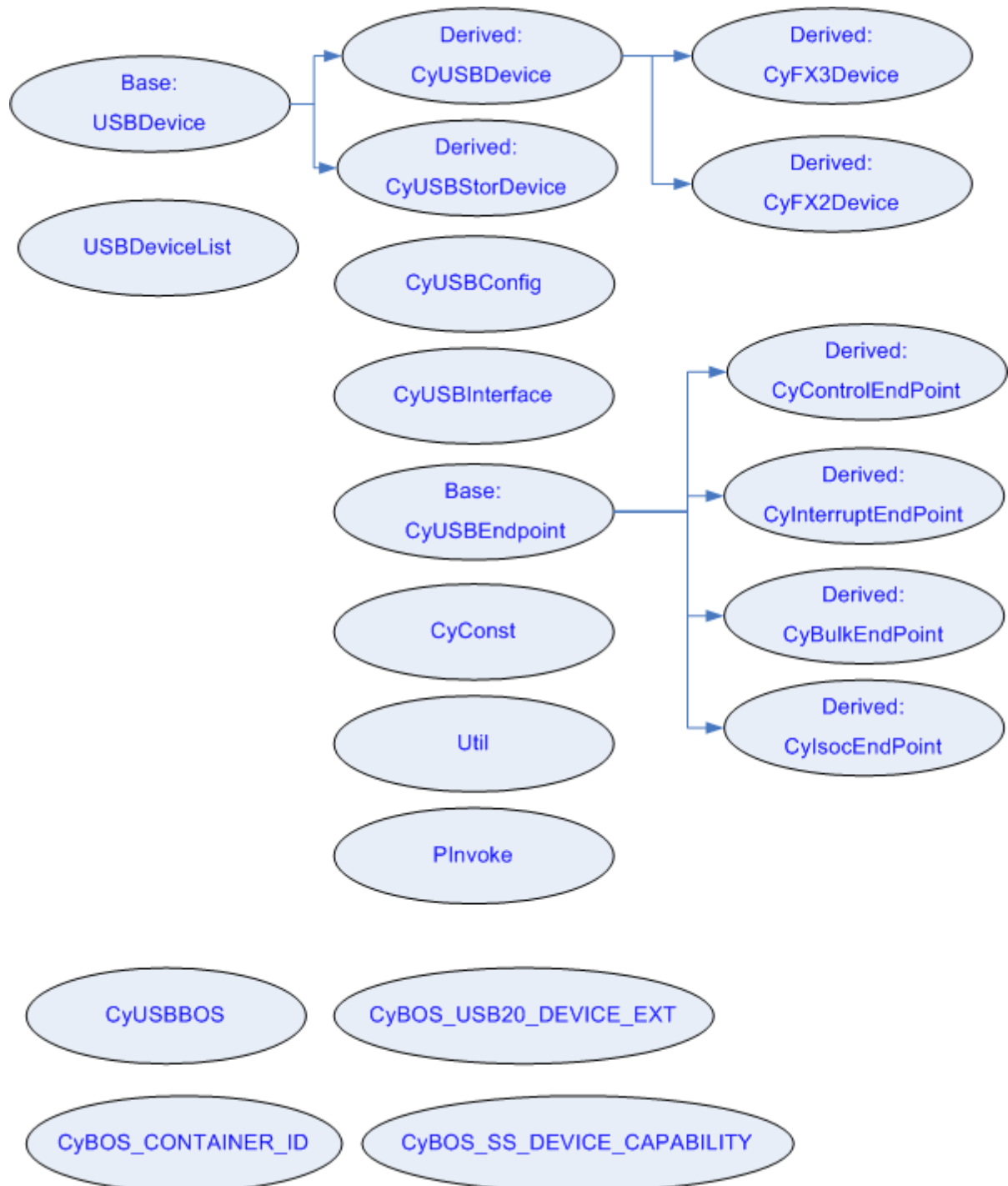
    void usbDevices_DeviceRemoved(object sender, EventArgs e)
    {
        USBEventArgs usbEvent = e as USBEventArgs;
        StatusLabel.Text = usbEvent.FriendlyName + " removed.";
    }

    void usbDevices_DeviceAttached(object sender, EventArgs e)
    {
        USBEventArgs usbEvent = e as USBEventArgs;
        StatusLabel.Text = usbEvent.Device.FriendlyName + " connected.";
    }
}
```



## 2 CyUSB Library Class Hierarchy

The class hierarchy diagram shown below illustrates the C# CyUSB DLL interface classes.



## 3 New Features

### New Features

[Top](#) [Previous](#) [Next](#)

#### Description

This section contains additional features that are found in recent releases of CyUSB.Net.

The current list of new features is as follows:

- [USB3.0 Support Overview](#)

## 3.1 USB3.0 Support Overview

### USB3.0 Support Overview

[Top](#) [Previous](#) [Next](#)

#### Description

The Binary Device Object Store(BOS) descriptor defines a root descriptor that is similar to the configuration descriptor and a base descriptor for accessing a family of related descriptors. A host can read the wTotalLength field of the BOS descriptor to find the length of the device level descriptor set.

#### API

All BOS support APIs are incorporated in the [CyUSBDevice](#) class.

[GetBosDescriptor\(\)](#)

[GetBosContainerIDDescriptor\(\)](#)

[GetBosSSCapabilityDescriptor\(\)](#)

[GetBosUSB20DeviceExtensionDesc\(\)](#)

#### Data Structure

All BOS data structure definitions are defined in the [CyUSB](#) namespace.

[USB\\_BOS\\_DESCRIPTOR](#)

[USB\\_BOS\\_CONTAINER\\_ID](#)

[USB\\_BOS\\_SS\\_DEVICE\\_CAPABILITY](#)

[USB\\_BOS\\_USB20\\_DEVICE\\_EXTENSION](#)

#### Classes

All BOS class definitions are defined in the [CyUSB](#) namespace.

[CyBOS\\_CONTAINER\\_ID](#)

[CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#)

[CyBOS\\_USB20\\_DEVICE\\_EXT](#)

#### Device Speed

Super Speed variable is defined in the [CyUSBDevice](#) class.

[bSuperSpeed](#)

#### SuperSpeed Endpoint Companion descriptor

All Superspeed endpoint companion descriptor data variable definitions are incorporated in the [CyUSBEndPoint](#). The variables shown below will be initialized with zero if device is USB2.0 and set the superspeed endpoint companion descriptor values for USB3.0 device.

[SSDscLen](#)

[SSDscType](#)

[SSBytePerInterval](#)

[SSBmAttribute](#)

[SSMaxBurst](#)

#### Firmware Download

CyUSB library provides API to download the firmware image to FX3 hardware.

[CyFX3Device](#)

NOTE : Please note that this library does not support USB3.0 bulk streams.

## 4 CyUSB

namespace **CyUSB**

[Top](#) [Previous](#) [Next](#)

### Description

CyUSB is the .net namespace that defines all the classes in the *CyUSB.dll* library.

To use the library, you will need to declare the namespace in any source files that reference the CyUSB classes as shown here:

#### **Visual Basic.net**

```
Imports CyUSB
```

#### **Visual C#**

```
using CyUSB;
```

#### **Visual C++ (Win Forms App)**

```
using namespace CyUSB;
```

#### **Visual J#**

```
import CyUSB.*;
```

In addition, you will need to add the *CyUSB.dll* to the references folder of your project.

## 4.1 CyBOS\_CONTAINER\_ID

public class **CyBOS\_CONTAINER\_ID**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The CyBOS\_CONTAINER\_ID class represents the Container ID of a USB3.0 device.

An object of this class is instantiated while parsing the BOS configuration. The CyUSBBOS class references this instance.

The following example code shows usage of the CyBOS\_CONTAINER\_ID class in an application.

### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;  
  
string text = myDev.USBBos.Container_ID.ToString();
```

Fills text with the following:

```
<CONTAINER ID>  
    DescriptorLength="20"  
    DescriptorType="16"  
    DeviceCapabilityType="4"  
    bmAttribute="0h"  
    ContainerID="00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 14 16"  
</CONTAINER ID>
```

### 4.1.1 Tree

```
public override System.Windows.Forms.TreeNode  
Tree { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyBOS\\_CONTAINER\\_ID](#)

#### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string returned by the [Container\\_ID](#) property.

The tree property does not have any child nodes.

The Tag property of the returned TreeNode contains a reference to the [CyBOS\\_CONTAINER\\_ID](#) object (*this*).

#### C# Example

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```

### 4.1.2 Length

```
public byte Length { get; }
```

Member of [CyUSB.CyBOS\\_CONTAINER\\_ID](#)

[Top](#) [Previous](#) [Next](#)

#### Description

Length contains the value of bLength field of Container ID descriptor.

### 4.1.3 DescriptorType

```
public byte DescriptorType { get; }  
Member of CyUSB.CyBOS\_CONTAINER\_ID
```

[Top](#) [Previous](#) [Next](#)**Description**

DescriptorType contains the value of bDescriptorType field of Container ID descriptor.



#### 4.1.4 DevCapabilityType

```
public byte DescriptorType { get; }  
Member of CyUSB.CyBOS\_CONTAINER\_ID
```

[Top](#) [Previous](#) [Next](#)

##### Description

DevCapabilityType contains the value of bDevCapabilityType field of Container ID descriptor.

### 4.1.5 Reserved

```
public byte Reserved { get; }  
Member of CyUSB.CyBOS\_CONTAINER\_ID
```

[Top](#) [Previous](#) [Next](#)**Description**

Reserved field of Container ID descriptor.

#### 4.1.6 ContainerID

```
public byte[] ContainerID { get; }  
Member of CyUSB.CyBOS\_CONTAINER\_ID
```

[Top](#) [Previous](#) [Next](#)

##### Description

ContainerID contains the value of ContainerID field of Container ID descriptor.

### 4.1.7 ToString()

public override string **ToString()**  
Member of [CyUSB.CyBOS\\_CONTAINER\\_ID](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ToString returns an XML string that represents a Container ID descriptor.

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);
```

```
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;
```

```
DescText.Text += MyDevice.Container_ID.ToString();
```

Fills DescText.Text with the following:

```
<CONTAINER ID>  
  DescriptorLength="20"  
  DescriptorType="16"  
  DeviceCapabilityType="4"  
  bmAttribute="0h"  
  ContainerID="00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 14 16"  
</CONTAINER ID>
```

## 4.2 CyBOS\_SS\_DEVICE\_CAPABILITY

public class **CyBOS\_SS\_DEVICE\_CAPABILITY**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The CyBOS\_SS\_DEVICE\_CAPABILITY class represents the BOS super speed device capability of a USB3.0 device.

An object of this class is instantiated while parsing the BOS configuration. The CyUSBBOS class references this instance.

The following example code shows the usage of the CyBOS\_SS\_DEVICE\_CAPABILITY class in an application.

### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;  
  
string text = myDev.USBBos.SS_DeviceCap.ToString();
```

Fills text with the following:

```
<SUPERSPEED USB>  
  DescriptorLength="10"  
  DescriptorType="16"  
  DeviceCapabilityType="3"  
  FunctionalitySupporte="0"  
  bmAttribute="0Eh"  
  U1Device Exit Latency="0"  
  U2Device Exit Latency="00h"  
</SUPERSPEED USB>
```

### 4.2.1 Tree

```
public override System.Windows.Forms.TreeNode  
Tree { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#)

#### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string returned by the [SS\\_DeviceCap](#) property.

The tree property does not have child nodes.

The Tag property of the returned TreeNode contains a reference to the [CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#) object (*this*).

#### C# Example

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```

## 4.2.2 Length

```
public byte Length { get; }
```

Member of [CyUSB.CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#)

[Top](#) [Previous](#) [Next](#)

### Description

Length contains the value of the bLength field of the SS Device capability descriptor.

### 4.2.3 DescriptorType

```
public byte DescriptorType { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)**Description**

DescriptorType contains the value of bDescriptorType field of the SS Device capability descriptor.



## 4.2.4 DevCapabilityType

```
public byte DevCapabilityType { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)

### Description

DevCapabilityType contains the value of bDevCapabilityType field of the SS Device capability descriptor.

### 4.2.5 bmAttribute

```
public byte bmAttribute { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)**Description**

bmAttribute contains the value of bmAttribute field of the SS Device capability descriptor.

## 4.2.6 SpeedsSupported

```
public ushort SpeedsSupported { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)

### Description

SpeedsSupported contains the value of wSpeedsSupported field of the SS Device capability descriptor.

## 4.2.7 FunctionalitySupport

```
public byte FunctionalitySupport { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)

### Description

FunctionalitySupport contains the value of FunctionalitySupport field of the SS Device capability descriptor.

## 4.2.8 U1DevExitLat

```
public byte U1DevExitLat { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)

### Description

U1DevExitLat contains the value of U1DevExitLat field (U1 device exit latency) of the SS Device capability descriptor.

### 4.2.9 U2DevExitLat

```
public ushort U2DevExitLat { get; }  
Member of CyUSB.CyBOS\_SS\_DEVICE\_CAPABILITY
```

[Top](#) [Previous](#) [Next](#)**Description**

U2DevExitLat contains the value of U2DevExitLat field (U2 device exit latency) of the SS Device capability descriptor.

## 4.2.10 ToString()

public override string **ToString()**  
Member of [CyUSB.CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#)

[Top](#) [Previous](#) [Next](#)

### Description

ToString returns an XML string that represents a USB Interface descriptor.

### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].ToString();
```

Fills DescText.Text with the following:

```
<SUPERSPEED USB>  
    DescriptorLength="10"  
    DescriptorType="16"  
    DeviceCapabilityType="3"  
    FunctionalitySupport="0"  
    bmAttribute="0Eh"  
    U1Device Exit Latency="0"  
    U2Device Exit Latency="00h"  
</SUPERSPEED USB>
```

## 4.3 CyBOS\_USB20\_DEVICE\_EXT

```
public class CyBOS_USB20_DEVICE_EXT
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyBOS\_USB20\_DEVICE\_EXT represents the BOS USB2.0 device extension capability of a USB3.0 device.

It is automatically instantiated while parsing the BOS configuration when the [USBBos](#) instance is created.

The following example code shows the usage of the CyBOS\_USB20\_DEVICE\_EXT class in an application.

### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;
```

```
DescText.Text += MyDevice.USBBos.USB20_DeviceExt.ToString();
```

Fills DescText.Text with the following:

```
<USB20 Device Extension>  
  DescriptorLength="7"  
  DescriptorType="16"  
  DeviceCapabilityType="2"  
  bmAttribute="00h"  
</USB20 Device Extension>
```



### 4.3.1 Tree

```
public override System.Windows.Forms.TreeNode  
Tree { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyBOS\\_USB20\\_DEVICE\\_EXT](#)

#### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string returned by the [USB20\\_DeviceExt](#) property.

The tree property does not have child nodes.

The Tag property of the returned TreeNode contains a reference to the [CyBOS\\_USB20\\_DEVICE\\_EXT](#) object (*this*).

#### C# Example

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```

### 4.3.2 Length

```
public byte Length { get; }  
Member of CyUSB.CyBOS\_USB20\_DEVICE\_EXT
```

[Top](#) [Previous](#) [Next](#)**Description**

Length contains the value of bLength field of the USB2.0 Device extension descriptor.

### 4.3.3 DescriptorType

```
public byte DescriptorType { get; }  
Member of CyUSB.CyBOS\_USB20\_DEVICE\_EXT
```

[Top](#) [Previous](#) [Next](#)

#### Description

DescriptorType contains the value of bDescriptorType field of the USB2.0 Device extension descriptor.

### 4.3.4 DevCapabilityType

```
public byte DevCapabilityType { get; }  
Member of CyUSB.CyBOS\_USB20\_DEVICE\_EXT
```

[Top](#) [Previous](#) [Next](#)

#### Description

DevCapabilityType contains the value of bDevCapabilityType field of the USB2.0 Device extension descriptor.

### 4.3.5 bmAttribute

```
public uint bmAttribute { get; }  
Member of CyUSB.CyBOS\_USB20\_DEVICE\_EXT
```

[Top](#) [Previous](#) [Next](#)

#### Description

bmAttribute contains the value of bmAttribute field of the USB2.0 Device extension descriptor.

### 4.3.6 ToString()

public override string **ToString()**  
Member of [CyUSB.CyBOS\\_USB20\\_DEVICE\\_EXT](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ToString returns an XML string that represents a USB Interface descriptor.

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
DescText.Text += MyDevice.USBBoS.USB20_DeviceExt.ToString();
```

Fills DescText.Text with the following:

```
<USB20 Device Extension>  
  DescriptorLength="7"  
  DescriptorType="16"  
  DeviceCapabilityType="2"  
  bmAttribute="00h"  
</USB20 Device Extension>
```

## 4.4 CyUSBBOS

public class **CyUSBBOS**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyUSBBOS represents a Binary device object store .

A CyUSBBOS object is automatically instantiated for USB3.0 device BOS descriptor when a [USBDeviceList](#) is created.

In the process of construction, CyUSBBOS creates instances for each capability. if device does not define specific capability then the valued of the instance will be NULL.

Following are the capability type.

[CyBOS\\_USB20\\_DEVICE\\_EXT](#),  
[CyBOS\\_CONTAINER\\_ID](#) and  
[CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#)

The following example code shows the usage of the CyUSBBOS class in an application.

### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;

string text = myDev.USBBos.ToString();
```

Fills text with the following:

```
<BOS>
  NumberOfDeviceCapability="02h"
  DescriptorType="15"
  DescriptorLength="5"
  TotalLength="22"
  <USB20 Device Extension>
    DescriptorLength="7"
    DescriptorType="16"
    DeviceCapabilityType="2"
    bmAttribute="00h"
  </USB20 Device Extension>
  <SUPERSPEED USB>
    DescriptorLength="10"
    DescriptorType="16"
    DeviceCapabilityType="3"
    FunctionalitySupport="0"
    bmAttribute="0Eh"
    U1Device Exit Latency="0"
    U2Device Exit Latency="00h"
  </SUPERSPEED USB>
</BOS>
```





#### 4.4.1 USB20\_DeviceExt

public [CyUSB.CyBOS\\_USB20\\_DEVICE\\_EXT](#)

[Top](#) [Previous](#) [Next](#)

USB20\_DeviceExt

Member of [CyUSB.CyUSBBOS](#)

##### Description

USB20\_DeviceExt is a [CyBOS\\_USB20\\_DEVICE\\_EXT](#) object which represents the USB2.0 device extension capability of BOS. It can be null if the device does not define this capability or if device is USB2.0.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.4.2 SS\_DeviceCap

public [CyUSB.CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#)  
SS\_DeviceCap  
Member of [CyUSB.CyUSBBOS](#)

[Top](#) [Previous](#) [Next](#)

##### Description

SS\_DeviceCap is a [CyBOS\\_SS\\_DEVICE\\_CAPABILITY](#) object which represents the USB3.0 device super speed capability of BOS. It can be null if the device does not define this capability or if the device is USB2.0.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

### 4.4.3 Container\_ID

public [CyUSB.CyBOS\\_CONTAINER\\_ID](#) Container\_ID  
Member of [CyUSB.CyUSBBOS](#)

[Top](#) [Previous](#) [Next](#)

#### Description

Container\_ID is a [CyBOS\\_CONTAINER\\_ID](#) object which represents the USB3.0 device container ID of BOS. It can be null if the device does not define this capability or if the device is USB2.0.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.4.4 Tree

```
public override System.Windows.Forms.TreeNode  
Tree { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBBOS](#)

##### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string 'BOS'.

The children of the node are comprised of the trees representing the [Container ID](#), [SS Device Capability](#) and [USB20 extension](#) of the device.

The Tag property of the returned TreeNode contains a reference to the CyUSBBOS object (*this*).

##### C# Example

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```

#### 4.4.5 Length

```
public byte Length { get; }  
Member of CyUSB.CyUSBBOS
```

[Top](#) [Previous](#) [Next](#)**Description**

This property returns length of the BOS descriptor.

#### 4.4.6 DescriptorType

```
public byte DescriptorType{ get; }  
Member of CyUSB.CyUSBBOS
```

[Top](#) [Previous](#) [Next](#)**Description**

DescriptorType contains value of the **bDescriptorType** field from the selected BOS descriptor.

#### 4.4.7 TotalLength

```
public ushort TotalLength { get; }  
Member of CyUSB.CyUSBBOS
```

[Top](#) [Previous](#) [Next](#)

##### Description

TotalLength contains value of the wTotalLength field from the selected BOS descriptor.

#### 4.4.8 NumDeviceCaps

```
public byte NumDeviceCaps { get; }  
Member of CyUSB.CyUSBBOS
```

[Top](#) [Previous](#) [Next](#)

##### Description

NumDeviceCaps contains value of the **bNumberOfDeviceCapability** field from the selected BOS descriptor.



## 4.4.9 ToString()

public override string **ToString()**  
Member of [CyUSB.CyUSBBOS](#)

[Top](#) [Previous](#) [Next](#)

### Description

ToString returns an XML string that represents a USB BOS descriptor.

### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
DescText.Text += MyDevice.USBBos.ToString();
```

Fills BosDescText.Text with the following:

```
<BOS>  
  NumberOfDeviceCapability="02h"  
  DescriptorType="15"  
  DescriptorLength="5"  
  TotalLength="22"  
  <USB20 Device Extension>  
    DescriptorLength="7"  
    DescriptorType="16"  
    DeviceCapabilityType="2"  
    bmAttribute="00h"  
  </USB20 Device Extension>  
  <SUPERSPEED USB>  
    DescriptorLength="10"  
    DescriptorType="16"  
    DeviceCapabilityType="3"  
    FunctionalitySupport="0"  
    bmAttribute="0Eh"  
    U1Device Exit Latency="0"  
    U2Device Exit Latency="00h"  
  </SUPERSPEED USB>  
</BOS>
```

## 4.5 CyBulkEndPoint

```
public class CyBulkEndPoint : CyUSB,  
CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyBulkEndPoint is a subclass of the [CyUSBEndPoint](#) abstract class. CyBulkEndPoint adds no methods or properties that are not already contained in its parent, [CyUSBEndPoint](#). Rather, it exists to provide a non-abstract implementation of the endpoint and for consistency of the object model. To learn more about the methods and properties of this class see [CyUSBEndPoint](#).

When an instance of CyUSBDevice is created, instances of this class are automatically created for all bulk endpoints as members of that class. Two such members of CyUSBDevice are [BulkInEndPoint](#) and [BulkOutEndPoint](#).

### C# Example

```
// Find a bulk IN endpoint in the EndPoints[] array  
CyBulkEndPoint BulkIn = null;  
  
// Create a list of devices served by CyUSB3.sys  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Just look at the first device in the list  
CyUSBDevice dev = usbDevices[0] as CyUSBDevice;  
  
foreach (CyUSBEndPoint ept in dev.EndPoints)  
    if (ept.bIn && (ept.Attributes == 2))  
        BulkIn = ept as CyBulkEndPoint;
```

## 4.6 CyConst

public static class **CyConst**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyConst is a static class that contains several constants used by the CyAPI library classes and that are useful as parameters to some of the class methods.

### 4.6.1 DEVICES\_CYUSB

public const byte **DEVICES\_CYUSB**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

#### Description

This constant is passed to the USBDeviceList constructor to select those USB devices that are served by the cyusb3.sys device driver or a custom derivative of that driver that has its own GUID.

The value of this constant is 0x01.

#### C# Example 1

```
// Create a list of devices served by cyusb3.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);

if (usbDevices.Count == 0) return;
```

#### C# Example 2

```
// Create a list of devices served by cyusb3.sys or usbstor.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB | CyConst.DEVICES_MSC);

if (usbDevices.Count == 0) return;
```

## 4.6.2 DEVICES\_HID

public const byte **DEVICES\_HID**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is passed to the USBDeviceList constructor to select USB Human Interface Devices.

The value of this constant is 0x04.

### C# Example 1

```
// Create a list of HID devices
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_HID);

if (usbDevices.Count == 0) return;
```

### C# Example 2

```
// Create a list of HID devices or devices served by cyusb3.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_HID | CyConst.DEVICES_CYUSB);

if (usbDevices.Count == 0) return;
```

### 4.6.3 DEVICES\_MSC

public const byte **DEVICES\_MSC**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

#### Description

This constant is passed to the USBDeviceList constructor to select USB Mass Storage Class devices that are served by the Windows usbstor.sys device driver.

The value of this constant is 0x02.

#### C# Example 1

```
// Create a list of Mass Storage Class devices
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_MSC);

if (usbDevices.Count == 0) return;
```

#### C# Example 2

```
// Create a list of Mass Storage Class devices or devices served by cyusb3.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_MSC | CyConst.DEVICES_CYUSB);

if (usbDevices.Count == 0) return;
```

#### 4.6.4 DIR\_FROM\_DEVICE

public const byte **DIR\_FROM\_DEVICE**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

##### Description

This constant is used to set the [Direction](#) property of a [CyControlEndPoint](#) object.

The value of DIR\_FROM\_DEVICE is 0x80.

When the [Direction](#) property of [CyControlEndPoint](#) is set to DIR\_FROM\_DEVICE, control transfers will move data from the USB device to the USB host (i.e. PC).

##### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_FROM_DEVICE;
    CtrlEndPt.RegCode     = 0xB0;
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 0;

    int len = 64;
    byte[] buf = new byte[len];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.6.5 DIR\_TO\_DEVICE

public const byte **DIR\_TO\_DEVICE**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is used to set the [Direction](#) property of a [CyControlEndPoint](#) object.

The value of DIR\_TO\_DEVICE is 0x00.

When the [Direction](#) property of [CyControlEndPoint](#) is set to DIR\_TO\_DEVICE, control transfers will move data from the USB host (i.e. PC) to the USB device.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice         = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.RegCode     = 0xC0;
    CtrlEndPt.Value       = 2;
    CtrlEndPt.Index       = 0;

    int len = 0;
    byte[] buf = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```



## 4.6.6 REQ\_CLASS

public const byte **REQ\_CLASS**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is used to set the [ReqType](#) property of a [CyControlEndPoint](#) object.

The value of REQ\_CLASS is 0x20.

When the [ReqType](#) property of [CyControlEndPoint](#) is set to REQ\_CLASS, the ReqCode parameter will be interpreted as a class-specific argument.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices       = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice          MyDevice         = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.ReqType     = CyConst.REQ_CLASS;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.ReqCode     = 0x06;           // Some class-specific request code
    CtrlEndPt.Value       = 3;
    CtrlEndPt.Index       = 1;

    int len = 0;
    byte[] buf           = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.6.7 REQ\_STD

public const byte **REQ\_STD**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is used to set the [ReqType](#) property of a [CyControlEndPoint](#) object.

The value of REQ\_STD is 0x00.

When the [ReqType](#) property of [CyControlEndPoint](#) is set to REQ\_STD, the ReqCode parameter will be interpreted as one of the standard requests.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.ReqType     = CyConst.REQ_STD;
    CtrlEndPt.Direction   = CyConst.DIR_FROM_DEVICE;
    CtrlEndPt.ReqCode     = 0x06;           // Get Descriptor Standard Request
    CtrlEndPt.Value       = 0x200;         // Configuration Descriptor
    CtrlEndPt.Index       = 0;

    int len = 256;
    byte[] buf          = new byte[len];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

### 4.6.8 REQ\_VENDOR

public const byte **REQ\_VENDOR**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

#### Description

This constant is used to set the [ReqType](#) property of a [CyControlEndPoint](#) object.

The value of REQ\_VENDOR is 0x40.

When the [ReqType](#) property of [CyControlEndPoint](#) is set to REQ\_VENDOR, the ReqCode parameter will be interpreted as a vendor-specific request.

#### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.ReqType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.ReqCode     = 0xB1;           // Some vendor-specific request code
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 1;

    int len = 0;
    byte[] buf           = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.6.9 SINGLE\_XFER\_LEN

public const byte **SINGLE\_XFER\_LEN**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is used to allocate a command buffer that will be passed in a call to the [BeginDataXfer](#) method of a [CyUSBEndPoint](#) object.

The value of SINGLE\_XFER\_LEN is 38.

### C# Example

```
unsafe static void function()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0x04B4, 0x4C54] as CyUSBDevice;
    CyBulkEndPoint InEndpt;

    if (MyDevice != null)
        InEndpt = MyDevice.BulkInEndPt;
    else
        return;

    if (InEndpt != null)
    {
        byte[] cmdBuf = new byte[CyConst.SINGLE_XFER_LEN];
        byte[] xferBuf = new byte[512];
        byte[] overLap = new byte[CyConst.OverlapSignalAllocSize];
        int len = (CyConst.SINGLE_XFER_LEN+512);

        fixed (byte* tmp0 = overLap)
        {
            OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
            ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
        }

        InEndpt.BeginDataXfer(ref cmdBuf, ref xferBuf, ref len, ref overLap);
    }
}
```

## 4.6.10 TGT\_DEVICE

public const byte **TGT\_DEVICE**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is used to set the [Target](#) property of a [CyControlEndPoint](#) object.

The value of TGT\_DEVICE is 0x00.

When the [Target](#) property of [CyControlEndPoint](#) is set to TGT\_DEVICE, the intended recipient of the request is the device.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.RegCode     = 0xB1;           // Some vendor-specific request code
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 1;

    int len = 0;
    byte[] buf           = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

### 4.6.11 TGT\_ENDPT

public const byte **TGT\_ENDPT**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

#### Description

This constant is used to set the [Target](#) property of a [CyControlEndPoint](#) object.

The value of TGT\_ENDPT is 0x02.

When the [Target](#) property of [CyControlEndPoint](#) is set to TGT\_ENDPT, the intended recipient of the request is the endpoint indicated by the Index field.

#### C# Example

```
CyControlEndPoint  CtrlEndPt      = null;

USBDeviceList      usbDevices     = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice        MyDevice      = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_ENDPT;
    CtrlEndPt.ReqType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.ReqCode     = 0xE0;           // Some vendor-specific request code
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 2;             // Request is for endpoint 2

    int len = 0;
    byte[] buf           = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.6.12 TGT\_INTFC

public const byte **TGT\_INTFC**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

This constant is used to set the [Target](#) property of a [CyControlEndPoint](#) object.

The value of TGT\_INTFC is 0x01.

When the [Target](#) property of [CyControlEndPoint](#) is set to TGT\_INTFC, the intended recipient of the request is the interface indicated by the Index field.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice         = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_INTFC;
    CtrlEndPt.ReqType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction  = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.ReqCode     = 0x20;           // Some vendor-specific request code
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 1;             // Request is for interface 1

    int len = 0;
    byte[] buf          = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

### 4.6.13 TGT\_OTHER

public const byte **TGT\_OTHER**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

#### Description

This constant is used to set the [Target](#) property of a [CyControlEndPoint](#) object.

The value of TGT\_OTHER is 0x03.

When the [Target](#) property of [CyControlEndPoint](#) is set to TGT\_OTHER, the intended recipient of the request is other than the Device, Interface or Endpoint.

#### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices       = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice        = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_OTHER;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.RegCode     = 0x20;           // Some vendor-specific request code
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 1;

    int len = 0;
    byte[] buf           = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```



#### 4.6.14 INFINITE

public const uint **INFINITE**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

##### Description

This constant may be passed to the [WaitForSingleObject](#) method of [PInvoke](#) to cause that function to wait forever for the designated event to occur.

The value of INFINITE is 0xFFFFFFFF.

##### C# Example

```
unsafe static void function()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0x04B4, 0x4C54] as CyUSBDevice;
    CyBulkEndPoint InEndpt;
    byte[] overLap = new byte[CyConst.OverlapSignalAllocSize];

    if (MyDevice != null)
        InEndpt = MyDevice.BulkInEndPt;
    else
        return;

    fixed (byte* tmp0 = overLap)
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        bool retval = InEndpt.WaitForXfer(ovLapStatus->hEvent, (uint)500);
        if (!retval)
        {
            InEndpt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, CyConst.INFINITE);
        }
    }
}
```

#### 4.6.15 FX3\_FWDWNLOAD\_MEDIA\_TYPE

public enum FX3\_FWDWNLOAD\_MEDIA\_TYPE

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

##### **Description**

This is enumerator for the types of Firmware media.

It defined following media types

RAM - Download firmware to Ram.

I2CE2PROM - Download firmware to I2C E2PROM.

SPIFLASH - Download firmware to SPI FLASH.

#### 4.6.16 FX3\_FWDWNLOAD\_ERROR\_CODE

public enum FX3\_FWDWNLOAD\_ERROR\_CODE

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

##### Description

This enumerator defines following firmware download error codes.

SUCCESS-Firmware download successful

FAILED- Firmware download failed

INVALID\_MEDIA\_TYPE-Given Input Media type is not supported

INVALID\_FWSIGNATURE- Invalid Firmware Signature

DEVICE\_CREATE\_FAILED-Device Open failed

INCORRECT\_IMAGE\_LENGTH-Firmware image length is incorrect

INVALID\_FILE- Invalid file

SPILASH\_ERASE\_FAILED- SPI erase operation failed

CORRUPT\_FIRMWARE\_IMAGE\_FILE - Corrupt Firmware image file

I2CE2PROM\_UNKNOWN\_I2C\_SIZE - Unknown I2CE2PROM size, Unknown value parsed from 2nd Bytes of IMG file

## 4.7 CyControlEndPoint

public class **CyControlEndPoint** : [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyControlEndPoint is a subclass of the [CyUSBEndPoint](#) abstract class.

All USB devices have at least one Control endpoint, endpoint zero. Whenever an instance of [CyUSBDevice](#) is created, a member instance of CyControlEndPoint, called [ControlEndPt](#), is also instantiated. Normally, you will use this [ControlEndPt](#) member of CyUSBDevice to perform all your Control endpoint data transfers.

The CyControlEndPoint class contains 6 properties that should be set before performing a data transfer. These are:

[Target](#)  
[ReqType](#)  
[Direction](#)  
[ReqCode](#)  
[Value](#)  
[Index](#)

Control endpoint transfers are limited to 4K (4096) bytes.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices       = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice          StreamDevice     = usbDevices["Cy Stream Device"] as CyUSBDevice;

if (StreamDevice != null)
    CtrlEndPt = StreamDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.ReqType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction  = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.ReqCode    = 0xB1;           // Some vendor-specific request code
    CtrlEndPt.Value      = 0;
    CtrlEndPt.Index      = 1;

    int len = 0;
    byte[] buf      = new byte[1];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

### 4.7.1 Read()

```
public bool Read ( ref byte[] buf, ref int len )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

#### Description

Read( ) sets the CyControlEndPoint [Direction](#) member to DIR\_FROM\_DEVICE and then calls [CyControlEndPoint.XferData](#)( ).

The **buf** parameter holds the data bytes read from the device.

The **len** parameter tells how many bytes are to be read and must not exceed 4K (4096) bytes.

Returns **true** if the read operation was successful.

Passes-back the actual number of bytes transferred in the **len** parameter.

#### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPoint;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.RegCode     = 0xC0;
    CtrlEndPt.Value       = 2;
    CtrlEndPt.Index       = 0;

    int len              = 128;
    byte[] buf           = new byte[len];

    CtrlEndPt.Read(ref buf, ref len);

    bool success = (len > 0);
}
```

## 4.7.2 Write( )

```
public bool Write ( ref byte[] buf, ref System.
Int32 len )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

Write( ) sets the CyControlEndPoint [Direction](#) member to DIR\_TO\_DEVICE and then calls [CyUSBEndPoint.XferData](#)( ).

The **buf** parameter contains the data bytes that will be written to the device.

The **len** parameter tells how many bytes are to be written to the device and must not exceed 4K (4096) bytes.

Returns **true** if the write operation was successful.

Passes-back the actual number of bytes transferred in the **len** parameter.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice          MyDevice        = usbDevices[0] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.RegCode     = 0xC2;
    CtrlEndPt.Value       = 2;
    CtrlEndPt.Index       = 0;

    int len              = 128;
    byte[] buf           = new byte[len];

    CtrlEndPt.Write(ref buf, ref len);

    bool success = (len == 128);
}
```

### 4.7.3 XferData( )

```
unsafe public new bool XferData ( ref byte[] buf,
ref int len )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

#### Description

The XferData method of [CyControlEndPoint](#) hides the [XferData](#) method inherited from the [CyUSBEndPoint](#) class.

Control transfers require 6 parameters that are not needed for bulk, isoc, or interrupt transfers. These are:

[Target](#)  
[ReqType](#)  
[Direction](#)  
[ReqCode](#)  
[Value](#)  
[Index](#)

Be sure to set the value of these [CyControlEndPoint](#) members before invoking the XferData method.

Control endpoint transfers are limited to 4K (4096) bytes.

#### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPt.RegCode     = 0xC0;
    CtrlEndPt.Value       = 2;
    CtrlEndPt.Index       = 0;

    int len              = 128;
    byte[] buf           = new byte[len];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.7.4 Direction

```
public byte Direction { set; get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

The Direction property determines whether data is transferred from the host to the device or from the device to the host.

Legitimate values for the Direction member are [DIR\\_TO\\_DEVICE](#) and [DIR\\_FROM\\_DEVICE](#).

Unlike Bulk, Interrupt and ISOC endpoints, which are uni-directional (either IN or OUT), the Control endpoint is bi-directional. It can be used to send data to the device or read data from the device. So, the direction of the transaction is one of the fundamental parameters required for each Control transfer.

Direction is automatically set to [DIR\\_TO\\_DEVICE](#) by the [Write\(\)](#) method. It is automatically set to [DIR\\_FROM\\_DEVICE](#) by the [Read\(\)](#) method.

### C# Example

```
CyControlEndPoint    CtrlEndPoint    = null;

USBDeviceList        usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice          MyDevice        = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPoint = MyDevice.ControlEndPoint;

if (CtrlEndPoint != null)
{
    CtrlEndPoint.Target      = CyConst.TGT_DEVICE;
    CtrlEndPoint.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPoint.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPoint.RegCode     = 0xC0;
    CtrlEndPoint.Value       = 2;
    CtrlEndPoint.Index       = 0;

    int len = 0;
    byte[] buf = new byte[1];

    CtrlEndPoint.XferData(ref buf, ref len);
}
```



## 4.7.5 Index

public ushort **Index** { set; get; }

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

The Index property indicates the recipient endpoint number if the [Target](#) property is set to [TGT\\_ENDPT](#). Or, if the Target property is set to [TGT\\_INTFC](#), it indicates the recipient interface number.

In other cases, the Index field often holds parameters for the commands that are being sent through the Control endpoint.

### C# Example

```
CyControlEndPoint    CtrlEndPoint    = null;

USBDeviceList        usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice       = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPoint = MyDevice.ControlEndPoint;

if (CtrlEndPoint != null)
{
    CtrlEndPoint.Target    = CyConst.TGT_ENDPT;
    CtrlEndPoint.RegType   = CyConst.REQ_VENDOR;
    CtrlEndPoint.Direction = CyConst.DIR_TO_DEVICE;
    CtrlEndPoint.RegCode   = 0xE0;           // Some vendor-specific request code
    CtrlEndPoint.Value     = 0;
    CtrlEndPoint.Index     = 2;           // Request is for endpoint 2

    int len = 0;
    byte[] buf    = new byte[1];

    CtrlEndPoint.XferData(ref buf, ref len);
}
```

## 4.7.6 ReqCode

```
public byte ReqCode { set; get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

The ReqCode property indicates, to the USB device, a particular function or command that the device should perform.

When the [ReqType](#) property is [REQ\\_STD](#), the possible values of ReqCode are documented in the USB 2.0 specification.

For [ReqType](#) == [REQ\\_VENDOR](#), the ReqCode will indicate a vendor-specific command code for the device.

### C# Example

```
CyControlEndPoint    CtrlEndPoint    = null;

USBDeviceList        usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice          MyDevice        = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPoint = MyDevice.ControlEndPoint;

if (CtrlEndPoint != null)
{
    CtrlEndPoint.Target      = CyConst.TGT_DEVICE;
    CtrlEndPoint.ReqType     = CyConst.REQ_STD;
    CtrlEndPoint.Direction  = CyConst.DIR_FROM_DEVICE;
    CtrlEndPoint.ReqCode = 0x06;      // Get Descriptor Standard Request
    CtrlEndPoint.Value      = 0x200;  // Configuration Descriptor
    CtrlEndPoint.Index      = 0;

    int len = 256;
    byte[] buf = new byte[len];

    CtrlEndPoint.XferData(ref buf, ref len);
}
```

## 4.7.7 ReqType

```
public byte ReqType { set; get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

The ReqType property indicates, to the USB device, how it should interpret the ReqCode field of the control transfer.

When the ReqType property is [REQ\\_STD](#), the possible values of ReqCode are documented in the USB 2.0 specification.

When the ReqType property is [REQ\\_CLASS](#), the possible values of ReqCode are documented in the specification for the device's USB Class.

When the ReqType property is [REQ\\_VENDOR](#), the ReqCode will indicate a vendor-specific command code for the device.

### C# Example

```
CyControlEndPoint    CtrlEndPt        = null;

USBDeviceList        usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice           MyDevice          = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPoint;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.ReqType     = CyConst.REQ_STD;
    CtrlEndPt.Direction   = CyConst.DIR_FROM_DEVICE;
    CtrlEndPt.ReqCode     = 0x06;           // Get Descriptor Standard Request
    CtrlEndPt.Value       = 0x200;         // Configuration Descriptor
    CtrlEndPt.Index       = 0;

    int len = 256;
    byte[] buf = new byte[len];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.7.8 Target

public byte **Target** { set; get; }

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

The Target property indicates to which level of the USB device the control transfer is directed. It represents the Recipient bitfield of the bmRequestType field of a USB Device Request as documented in the USB 2.0 specification.

Legitimate values for the Target member are [TGT\\_DEVICE](#), [TGT\\_INTFC](#), [TGT\\_ENDPT](#) and [TGT\\_OTHER](#).

### C# Example

```
CyControlEndPoint  CtrlEndPoint      = null;

USBDeviceList      usbDevices        = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice        MyDevice          = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPoint = MyDevice.ControlEndPoint;

if (CtrlEndPoint != null)
{
    CtrlEndPoint.Target      = CyConst.TGT_ENDPT;
    CtrlEndPoint.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPoint.Direction   = CyConst.DIR_TO_DEVICE;
    CtrlEndPoint.RegCode     = 0xE0;           // Some vendor-specific request code
    CtrlEndPoint.Value       = 0;
    CtrlEndPoint.Index       = 2;           // Request is for endpoint 2

    int len = 0;
    byte[] buf = new byte[1];

    CtrlEndPoint.XferData(ref buf, ref len);
}
```

## 4.7.9 Value

```
public ushort Value { set; get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyControlEndPoint](#)

### Description

The value field often holds parameters for the requests that are being sent through the Control endpoint.

### C# Example

```
CyControlEndPoint  CtrlEndPt      = null;

USBDeviceList      usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice         MyDevice       = usbDevices[0x04B4,0x4C54] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPoint;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_STD;
    CtrlEndPt.Direction   = CyConst.DIR_FROM_DEVICE;
    CtrlEndPt.RegCode     = 0x06;           // Get Descriptor Standard Request
    CtrlEndPt.Value       = 0x200;         // Specifies the Configuration Descriptor
    CtrlEndPt.Index       = 0;

    int len = 256;
    byte[] buf      = new byte[len];

    CtrlEndPt.XferData(ref buf, ref len);
}
```

## 4.8 CyFX2Device

public class **CyFX2Device** : [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyFX2Device extends the functionality of [CyUSBDevice](#) by adding three methods specific to the Cypress FX2 family of programmable USB chips.

Note that any CyUSBDevice in a [USBDeviceList](#) object is also capable of being cast into a CyFX2Device. However, only those that represent actual FX2 devices will function properly when the [LoadEEPROM](#), [LoadRAM](#) and [Reset](#) methods of CyFX2Device are invoked.

The behavior of non-FX2 devices, in response to these methods, is undefined.

### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyFX2Device    fx2 = usbDevices["Cy Stream Device"] as CyFX2Device;  
  
bool bResult = fx2.LoadEEPROM("CustomFW.iic");
```

## 4.8.1 LoadEEPROM

public bool **LoadEEPROM**(string *fwFile*)

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyFX2Device](#)

### Description

The LoadEEPROM method of CyFX2Device writes the contents of an **.iic** firmware image file to an EEPROM attached to an FX2 device and verifies that the image was successfully written by reading back the EEPROM contents.

The file containing the firmware image is named in the *fwFile* parameter.

LoadEEPROM returns *true* if the operation succeeds and *false* otherwise.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void ProgE2Item_Click(object sender, EventArgs e)
{
    TreeNode selNode = DeviceTreeView.SelectedNode;

    if (selNode == null)
    {
        MessageBox.Show("Select an FX2 device in the device tree.", "Non-FX2 device selected");
        return;
    }

    // Climb to the top of the tree
    while (selNode.Parent != null)
        selNode = selNode.Parent;

    CyFX2Device fx2 = selNode.Tag as CyFX2Device;

    if (fx2 == null)
        MessageBox.Show("Select an FX2 device in the device tree.", "Non-FX2 device selected");
    else
    {
        if (FOpenDialog.ShowDialog() == DialogResult.OK)
        {
            bool bResult = false;

            if (sender == ProgE2Item)
            {
                StatLabel.Text = "Programming EEPROM of " + selNode.Text;
                Refresh();
                bResult = fx2.LoadEEPROM(FOpenDialog.FileName);
            }
            else
            {
                StatLabel.Text = "Programming RAM of " + selNode.Text;
                Refresh();
                bResult = fx2.LoadRAM(FOpenDialog.FileName);
            }

            StatLabel.Text = "Programming " + (bResult ? "succeeded." : "failed.");
            Refresh();
        }
    }
}
```

```
}  
}
```



## 4.8.2 LoadRAM

public bool **LoadRAM**(string *fwFile*)

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyFX2Device](#)

### Description

The LoadRAM method of CyFX2Device writes the contents of an **.iic** or a **.hex** firmware image file to the internal RAM of an FX2 device and, then, re-starts the device, running the new downloaded firmware.

The file containing the firmware image is named in the *fwFile* parameter.

LoadRAM returns *true* if the operation succeeds and *false* otherwise.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void ProgE2Item_Click(object sender, EventArgs e)
{
    TreeNode selNode = DeviceTreeView.SelectedNode;

    if (selNode == null)
    {
        MessageBox.Show("Select an FX2 device in the device tree.", "Non-FX2 device selected");
        return;
    }

    // Climb to the top of the tree
    while (selNode.Parent != null)
        selNode = selNode.Parent;

    CyFX2Device fx2 = selNode.Tag as CyFX2Device;

    if (fx2 == null)
        MessageBox.Show("Select an FX2 device in the device tree.", "Non-FX2 device selected");
    else
    {
        if (FOpenDialog.ShowDialog() == DialogResult.OK)
        {
            bool bResult = false;

            if (sender == ProgE2Item)
            {
                StatLabel.Text = "Programming EEPROM of " + selNode.Text;
                Refresh();
                bResult = fx2.LoadEEPROM(FOpenDialog.FileName);
            }
            else
            {
                StatLabel.Text = "Programming RAM of " + selNode.Text;
                Refresh();
                bResult = fx2.LoadRAM(FOpenDialog.FileName);
            }

            StatLabel.Text = "Programming " + (bResult ? "succeeded." : "failed.");
            Refresh();
        }
    }
}
```

```
}  
}
```

### 4.8.3 Reset

```
public void Reset(int hold)
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyFX2Device](#)

#### Description

The Reset method of CyFX2Device halts or starts the FX2 chip.

The *hold* parameter determines the effect of the Reset command.

hold == 0 causes the FX2 to resume execution.

hold == 1 halts the chip.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void HaltItem_Click(object sender, EventArgs e)
{
    TreeNode selNode = DeviceTreeView.SelectedNode;

    if (selNode == null)
    {
        MessageBox.Show("Select an FX2 device in the device tree.", "Non-FX2 device selected");
        return;
    }

    // Climb to the top of the tree
    while (selNode.Parent != null)
        selNode = selNode.Parent;

    CyFX2Device fx2 = selNode.Tag as CyFX2Device;

    if (fx2 == null)
        MessageBox.Show("Select an FX2 device in the device tree.", "Non-FX2 device selected");
    else
    {
        if (sender == HaltItem)
            fx2.Reset(1);
        else
            fx2.Reset(0);
    }
}
```

## 4.9 CyFX3Device

public class **CyFX3Device** : [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

Please note that APIs provided by this class will only work with the FX3 boot devices. The behaviour of each API is undefined for FX3 non-boot devices.

CyFX3Device extends the functionality of [CyUSBDevice](#) by adding methods to download firmware to Cypress FX3 boot devices.

Note that any CyUSBDevice in a [USBDeviceList](#) object is also capable of being cast into a CyFX3Device. However, only those that represent actual FX3 boot devices will function properly when the [DownloadFw](#) methods of CyFX3Device is invoked.

If your device does not support FX3 boot loader then please use CyUSBDevice class instead of CyFX3Device class.

### C# Example

Get instance for FX3-boot device.

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
```

```
CyFX3Device fx3 = usbDevices["Cy FX3 Device"] as CyFX3Device;
```

```
FX3_FWDNLOAD_ERROR_CODE result = fx3.DownloadFw ("CustomFW.img", FX3_FWDNLOAD_MEDIA_TYPE.  
RAM);
```

Get instance for FX3 non-boot device.

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
```

```
CyUSBDevice fx3 = usbDevices["Cy FX3 Device"] as CyUSBDevice;
```

### 4.9.1 DownloadFw()

```
public FX3_FWDWNLOAD_ERROR_CODE DownloadFw(string filename,
FX3_FWDWNLOAD_MEDIA_TYPE enMediaType)
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyFX3Device](#)

#### Description

The DownloadFw method of CyFX3Device allows the user to download firmware to various media such as RAM, I2C E2PROM and SPI FLASH.

The file name of the firmware file is passed as the first parameter to the API. The file must be a \*.img to keep this operation from failing.

The second parameter defines the Media Type using members of [FX3\\_FWDWNLOAD\\_MEDIA\\_TYPE](#)

The API returns a [FX3\\_FWDWNLOAD\\_ERROR\\_CODE](#) return code. User can call the [GetFwErrorString](#) API to get the error code string.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void ProgFx3Item_Click(object sender, EventArgs e)
{
    TreeNode selNode = DeviceTreeView.SelectedNode;

    if (selNode == null)
    {
        MessageBox.Show("Select an FX3 device in the device tree.", "Non-FX3 device selected");
        return;
    }

    // Climb to the top of the tree
    while (selNode.Parent != null)
        selNode = selNode.Parent;

    CyFX3Device fx3 = selNode.Tag as CyFX3Device;

    if (fx3 == null)
        MessageBox.Show("Select an FX3 device in the device tree.", "Non-FX3 device selected");
    else
    {
        if (FOpenDialog.ShowDialog() == DialogResult.OK)
        {
            FX3_FWDWNLOAD_ERROR_CODE enmResult = FX3_FWDWNLOAD_ERROR_CODE.SUCCESS;
            if (sender == ProgE2Item)
            {
                StatLabel.Text = "Programming RAM of " + selNode.Text;
                Refresh();
                enmResult = fx3.DownloadFw(FOpenDialog.FileName, FX3_FWDWNLOAD_MEDIA_TYPE.RAM);
            }

            StatLabel.Text = "Programming " + fx3.GetFwErrorString(enmResult);
            Refresh();
        }
    }
}
```

## 4.9.2 IsBootLoaderRunning( )

```
public bool IsBootLoaderRunning()
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyFX3Device](#)

### Description

The IsBootLoaderRunning function sends a vendor command to check boot loader status. If boot loader is running then it will return true otherwise false.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void CheckBootLaoderStatus_Click(object sender, EventArgs e)
{
    TreeNode selNode = DeviceTreeView .SelectedNode;

    if (selNode == null)
    {
        MessageBox.Show ("Select an FX3 device in the device tree.", "Non-FX3 device selected");
        return;
    }

    // Climb to the top of the tree
    while (selNode.Parent != null)
        selNode = selNode.Parent;

    CyFX3Device fx3 = selNode.Tag as CyFX3Device;

    if (fx3 == null)
        MessageBox.Show ("Select an FX3 device in the device tree.", "Non-FX3 device selected");
    else
    {
        bool bResult = fx3->IsBootLoaderRunning();
        StatLabel.Text = "Bootloader is " + (bResult ? "Running." : "Not Running.");
        Refresh();
    }
}
```

### 4.9.3 GetFwErrorString( )

```
public System.String GetFwErrorString(  
FX3\_FWDWNLOAD\_ERROR\_CODE eFwErrorCode)
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyFX3Device](#)

#### Description

This function returns the firmware error code string for the given input parameter [FX3\\_FWDWNLOAD\\_ERROR\\_CODE](#).

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void ProgFx3Item_Click(object sender, EventArgs e)
{
    TreeNode selNode = DeviceTreeView.SelectedNode;

    if (selNode == null)
    {
        MessageBox.Show("Select an FX3 device in the device tree.", "Non-FX3 device selected");
        return;
    }

    // Climb to the top of the tree
    while (selNode.Parent != null)
        selNode = selNode.Parent;

    CyFX3Device fx3 = selNode.Tag as CyFX3Device;

    if (fx3 == null)
        MessageBox.Show("Select an FX3 device in the device tree.", "Non-FX3 device selected");
    else
        if (FOpenDialog.ShowDialog() == DialogResult.OK)
        {
            FX3_FWDWNLOAD_ERROR_CODE enmResult = FX3_FWDWNLOAD_ERROR_CODE.SUCCESS;
            if (sender == ProgE2Item)
            {
                StatLabel.Text = "Programming RAM of " + selNode.Text;
                Refresh();
                enmResult = fx3.DownloadFw(FOpenDialog.FileName, FX3_FWDWNLOAD_MEDIA_TYPE.RAM);
            }

            StatLabel.Text = "Programming " + fx3.GetFwErrorString(enmResult);
            Refresh();
        }
}
```

## 4.10 CyInterruptEndPoint

```
public class CyInterruptEndPoint : CyUSB,  
CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyInterruptEndPoint is a subclass of the [CyUSBEndPoint](#) abstract class. CyInterruptEndPoint adds no methods or properties that are not already contained in its parent, [CyUSBEndPoint](#). Rather, it exists to provide a non-abstract implementation of the endpoint and for consistency of the object model. To learn more about the methods and properties of this class see [CyUSBEndPoint](#).

Instances of this class are automatically created when a [CyUSBDevice](#) object is instantiated for a device that exposes one or more interrupt endpoints. Two such members of [CyUSBDevice](#) are [InterruptInEndPt](#) and [InterruptOutEndPt](#).

### C# Example

```
// Find an interrupt IN endpoint in the EndPoints[] array  
CyInterruptEndPoint InterruptIn = null;  
  
// Create a list of devices served by cyusb3.sys  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Just look at the first device in the list  
CyUSBDevice dev = usbDevices[0] as CyUSBDevice;  
  
foreach (CyUSBEndPoint ept in dev.EndPoints)  
    if (ept.In && (ept.Attributes == 3))  
        InterruptIn = ept as CyInterruptEndPoint;
```



## 4.11 CylsocEndPoint

```
public class CyIsocEndPoint : CyUSB,  
    CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CylsocEndPoint is a subclass of the [CyUSBEndPoint](#) abstract class. This class exists to provide special [ISOC packet information](#) handling for Isochronous transfers.

Instances of CylsocEndPoint are automatically created when a [CyUSBDevice](#) object is instantiated for a device that exposes one or more ISOC endpoints. Two such members of [CyUSBDevice](#) are [IsocInEndPt](#) and [IsocOutEndPt](#).

### C# Example

```
// Find an isoc OUT endpoint in the EndPoints[] array  
CylsocEndPoint IsocOut = null;  
  
// Create a list of devices served by cyusb3.sys  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Just look at the first device in the list  
CyUSBDevice dev = usbDevices[0] as CyUSBDevice;  
  
foreach (CyUSBEndPoint ept in dev.EndPoints)  
    if (!ept.IsIn && (ept.Attributes == 1))  
        IsocOut = ept as CylsocEndPoint;
```

### 4.11.1 BeginDataXfer()

```
public override bool BeginDataXfer ( ref byte[]
singleXfer, ref byte[] buffer, ref int len, ref byte[]
ov )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyIsocEndPoint](#)

#### Description

BeginDataXfer is an advanced method for performing asynchronous IO. This method sets-up all the parameters for a data transfer, initiates the transfer, and immediately returns, not waiting for the transfer to complete.

You will usually want to use the synchronous [XferData](#) method rather than the asynchronous BeginDataXfer/WaitForXfer/FinishDataXfer approach.

Again, the use of BeginDataXfer, [WaitForXfer](#), and [FinishDataXfer](#) is the difficult way to transfer data to and from a USB device. This approach should only be used if it is imperative that you squeeze every last bit of throughput from the USB.

The code, below, utilizes the asynchronous methods to queue multiple transfers so as to keep the USB bandwidth fully utilized.

If user set the XMODE to BUFFERED mode for particular endpoint then user need to allocate singleXfer (the command buffer) with size of SINGLE\_XFER\_LEN and data buffer length.

This buffer will be passed to the singleXfer the first parameter of BeginDataXfer. This is the requirement specific to the BUFFERED mode only. The below sample example shows the usage of it.

In the below code, notice that the singleXfer parameter (cmdBufs byte array) must be large enough to accommodate the [ISO\\_PKT\\_INFO](#) data for all the packets. The needed size is calculated by calling [GetPktBlockSize](#).

#### Advanced C# Example

```
public unsafe void ListenThread()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0] as CyUSBDevice;
    int XferBytes = 0;

    if (MyDevice == null) return;

    CyIsocEndPoint InEndpt = MyDevice.IsocInEndPt;

    byte i = 0;

    int BufSz = InEndpt.MaxPktSize * 15;
    int QueueSz = 8;

    InEndpt.XferSize = BufSz;

    // Setup the queue buffers
    byte[][] cmdBufs = new byte[QueueSz][];
```

```

byte[][] xferBufs = new byte[QueueSz][];
byte[][] ovLaps = new byte[QueueSz][];
ISO_PKT_INFO[] pktInfos = new ISO_PKT_INFO[QueueSz];

for (i = 0; i < QueueSz; i++)
{
    cmdBufs[i] = new byte[CyConst.SINGLE_XFER_LEN + InEndpt.GetPktBlockSize(BufSz) + ((InEndpt.XferMode ==
XMODE.BUFFERED) ? BufSz : 0)];
    pktInfos[i] = new ISO_PKT_INFO[InEndpt.GetPktCount(BufSz)];
    xferBufs[i] = new byte[BufSz];
    ovLaps[i] = new byte[CyConst.OverlapSignalAllocSize];

    fixed (byte* tmp0 = ovLaps[i])
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
    }
}
// Pre-load the queue with requests
int len = BufSz;
for (i = 0; i < QueueSz; i++)
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

i = 0;
int Successes = 0;
int Failures = 0;
XferBytes = 0;

for (; i < 16; i++)
{
    fixed (byte* tmp0 = ovLaps[i])
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        if (!InEndpt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            InEndpt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);
        }
    }
    if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i], ref pktInfos[i]))
    {
        XferBytes += len;
        Successes++;

        // Add code to examine each ISO_PKT_INFO here
    }
    else
        Failures++;

    // Re-submit this buffer into the queue
    len = BufSz;
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);
    i++;
}
}

```

### 4.11.2 FinishDataXfer( )

```
public virtual bool FinishDataXfer ( ref byte[]
singleXfer, ref byte[] buffer, ref int len, ref byte[]
ov, ref CyAPI.ISO_PKT_INFO[] pktInfo )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyIsocEndPoint](#)

#### Description

FinishDataXfer is an advanced method for performing asynchronous IO. This method completes the data transfer that was initiated by the [BeginDataXfer](#) method.

You will usually want to use the synchronous [XferData](#) method rather than the asynchronous BeginDataXfer/WaitForXfer/FinishDataXfer approach.

Again, the use of [BeginDataXfer](#), [WaitForXfer](#), and FinishDataXfer is the difficult way to transfer data to and from a USB device. This approach should only be used if it is imperative that you squeeze every last bit of throughput from the USB.

The code, below, utilizes the asynchronous methods to queue multiple transfers so as to keep the USB bandwidth fully utilized.

In the below code, notice that the singleXfer parameter (cmdBufs byte array) must be large enough to accommodate the [ISO\\_PKT\\_INFO](#) data for all the packets. The needed size is calculated by calling [GetPktBlockSize](#).

#### Advanced C# Example

```
public unsafe void ListenThread()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0] as CyUSBDevice;
    int XferBytes = 0;

    if (MyDevice == null) return;

    CyIsocEndPoint InEndpt = MyDevice.IsocInEndPt;

    byte i = 0;

    int BufSz = InEndpt.MaxPktSize * 15;
    int QueueSz = 8;

    InEndpt.XferSize = BufSz;

    // Setup the queue buffers
    byte[][] cmdBufs = new byte[QueueSz][];
    byte[][] xferBufs = new byte[QueueSz][];
    byte[][] ovLaps = new byte[QueueSz][];
    ISO_PKT_INFO[] pktInfos = new ISO_PKT_INFO[QueueSz];

    for (i = 0; i < QueueSz; i++)
    {
        cmdBufs[i] = new byte[CyConst.SINGLE_XFER_LEN + InEndpt.GetPktBlockSize(BufSz) + ((InEndpt.XferMode ==
XMODE.BUFFERED) ? BufSz : 0)];
        pktInfos[i] = new ISO_PKT_INFO[InEndpt.GetPktCount(BufSz)];
    }
}
```

```

xferBufs[i] = new byte[BufSz];
ovLaps[i] = new byte[CyConst.OverlapSignalAllocSize];

fixed (byte* tmp0 = ovLaps[i])
{
    OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
    ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
}
}

// Pre-load the queue with requests
int len = BufSz;
for (i = 0; i < QueueSz; i++)
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

i = 0;
int Successes = 0;
int Failures = 0;
XferBytes = 0;

for (; i < 16; i++)
{
    fixed (byte* tmp0 = ovLaps[i])
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        if (!InEndpt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            InEndpt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);
        }
    }
    if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i], ref pktInfos[i]))
    {
        XferBytes += len;
        Successes++;

        // Add code to examine each ISO_PKT_INFO here
    }
    else
        Failures++;

    // Re-submit this buffer into the queue
    len = BufSz;
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);
    i++;
}
}

```

### 4.11.3 GetPktBlockSize( )

```
public int GetPktBlockSize ( int len )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyIsocEndPoint](#)

#### Description

GetPktBlockSize returns the combined size of all the [ISO\\_PKT\\_INFO](#) structures that would be needed for an isochronous data transfer of *len* bytes.

This number of packets needed for the transfer is a function of the CyIsocEndPoint's [MaxPktSize](#).

Note that this method is only needed when using the asynchronous [BeginDataXfer/WaitForXfer/FinishDataXfer](#) technique of transferring data. If you use the [XferData](#) method, this calculation is handled automatically for you.

#### Advanced C# Example

```
public unsafe void ListenThread()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0] as CyUSBDevice;
    int XferBytes = 0;

    if (MyDevice == null) return;

    CyIsocEndPoint InEndpt = MyDevice.IsocInEndPt;

    byte i = 0;

    int BufSz = InEndpt.MaxPktSize * 15;
    int QueueSz = 8;

    InEndpt.XferSize = BufSz;

    // Setup the queue buffers
    byte[][] cmdBufs = new byte[QueueSz][];
    byte[][] xferBufs = new byte[QueueSz][];
    byte[][] ovLaps = new byte[QueueSz][];
    ISO_PKT_INFO[] pktInfos = new ISO_PKT_INFO[QueueSz];

    for (i = 0; i < QueueSz; i++)
    {
        cmdBufs[i] = new byte[CyConst.SINGLE_XFER_LEN + InEndpt.GetPktBlockSize(BufSz) + ((InEndpt.XferMode ==
XMODE.BUFFERED) ? BufSz : 0)];
        pktInfos[i] = new ISO_PKT_INFO[InEndpt.GetPktCount(BufSz)];
        xferBufs[i] = new byte[BufSz];
        ovLaps[i] = new byte[CyConst.OverlapSignalAllocSize];

        fixed (byte* tmp0 = ovLaps[i])
        {
            OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
            ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
        }
    }

    // Pre-load the queue with requests
```

```
int len = BufSz;
for (i = 0; i < QueueSz; i++)
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

i = 0;
int Successes = 0;
int Failures = 0;
XferBytes = 0;

for (; i < 16; )
{
    fixed (byte* tmp0 = ovLaps[i])
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        if (!InEndpt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            InEndpt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);
        }
    }
    if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i], ref pktInfos[i]))
    {
        XferBytes += len;
        Successes++;

        // Add code to examine each ISO_PKT_INFO here
    }
    else
        Failures++;

    // Re-submit this buffer into the queue
    len = BufSz;
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);
    i++;
}
}
```

#### 4.11.4 GetPktCount( )

```
public int GetPktCount ( int len )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyIsocEndPoint](#)

##### Description

GetPktCount returns the number of [ISO\\_PKT\\_INFO](#) structures that would be needed for an isochronous data transfer of *len* bytes.

This number is a function of the CyIsocEndPoint's [MaxPktSize](#).

##### C# Example

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice      MyDevice      = usbDevices[0x04B4,0x1003] as CyUSBDevice;

if (MyDevice != null)
    if (MyDevice.IsocInEndPt != null)
    {
        int len = MyDevice.IsocInEndPt.MaxPktSize * 8;

        byte [] buf = new byte[len];

        ISO_PKT_INFO[] pktInfos = new ISO_PKT_INFO[MyDevice.IsocInEndPt.GetPktCount(len)];

        MyDevice.IsocInEndPt.XferSize = len;

        MyDevice.IsocInEndPt.XferData(ref buf, ref len, ref pktInfos);
    }
```



### 4.11.5 XferData( )

```
unsafe public bool XferData ( ref byte[] buf, ref
int len, ref CyUSB.ISO\_PKT\_INFO[] pktInfos)
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyIsocEndPoint](#)

#### Description

The XferData method sends or receives *len* bytes of data from / into *buf*. It performs synchronous (i.e. blocking) IO operations and does not return until the transaction completes or the endpoint's [TimeOut](#) has elapsed.

This implementation of XferData also fills a passed array of [ISO\\_PKT\\_INFO](#) structures.

Returns *true* if the transaction successfully completes before [TimeOut](#) has elapsed.

See also the [CyIsocEndPoint.XferData](#) method that does not pass back an array of [ISO\\_PKT\\_INFO](#) structures.

Note that for ISOC transfers, the buffer length and the endpoint's transfers size must be a multiple of 8 times the endpoint's MaxPktSize.

#### C# Example

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice      MyDevice      = usbDevices[0x04B4,0x1003] as CyUSBDevice;

if (MyDevice != null)
    if (MyDevice.IsocInEndPt != null)
    {
        int len = MyDevice.IsocInEndPt.MaxPktSize * 8;

        byte [] buf = new byte[len];

        ISO_PKT_INFO[] pktInfos = new ISO_PKT_INFO[MyDevice.IsocInEndPt.GetPktCount(len)];

        MyDevice.IsocInEndPt.XferSize = len;

        MyDevice.IsocInEndPt.XferData(ref buf, ref len, ref pktInfos);
    }
```

### 4.11.6 XferData( )

```
unsafe public override bool XferData ( ref byte[]  
buf, ref int len )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyIsocEndPoint](#)

#### Description

The XferData method sends or receives *len* bytes of data from / into *buf*. It performs synchronous (i. e. blocking) IO operations and does not return until the transaction completes or the endpoint's [TimeOut](#) has elapsed.

Returns *true* if the transaction successfully completes before [TimeOut](#) has elapsed.

See also the [CylsocEndPoint.XferData](#) method that passes back an array of [ISO\\_PKT\\_INFO](#) structures.

Note that for ISOC transfers, the buffer length and the endpoint's transfers size must be a multiple of 8 times the endpoint's MaxPktSize.

#### C# Example

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice      MyDevice      = usbDevices[0x04B4,0x1003] as CyUSBDevice;  
  
if (MyDevice != null)  
    if (MyDevice.IsocInEndPt != null)  
    {  
        int len = MyDevice.IsocInEndPt.MaxPktSize * 8;  
  
        byte [] buf = new byte[len];  
  
        MyDevice.IsocInEndPt.XferSize = len;  
  
        MyDevice.IsocInEndPt.XferData(ref buf, ref len);  
    }
```

## 4.12 CyUSBConfig

public class **CyUSBConfig**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

CyUSBConfig represents a USB device configuration descriptor. Such configurations have one or more interfaces each of which exposes one or more endpoints.

A CyUSBConfig object is automatically instantiated for each configuration of each device when a [USBDeviceList](#) is created.

In the process of construction, CyUSBConfig creates instances of [CyUSBInterface](#) for each interface exposed in the device's configuration descriptor. In turn, the [CyUSBInterface](#) class creates instances of [CyUSBEndPoint](#) for each endpoint descriptor contained in the interface descriptor. In this iterative fashion, the entire structure of Configs->Interfaces->EndPoints gets populated from a single construction of the CyUSBConfig class.

The following example code shows the use of the CyUSBConfig class in an application.

### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;

string text = myDev.USBCfgs[0].ToString();
```

Fills text with the following:

```
<CONFIGURATION>
  Configuration="0"
  ConfigurationValue="1"
  Attributes="0xA0"
  Interfaces="1"
  DescriptorType="2"
  DescriptorLength="9"
  TotalLength="135"
  MaxPower="50"
  <INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="0"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
  <ENDPOINT>
    Type="BULK"
```

```

        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="1"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x02"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"

```

```
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISO"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="4"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISO"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
```

```

AltSetting="5"
Class="0xFF"
Subclass="0x00"
Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="ISO"
    Direction="IN"
    Address="0x82"
    Attributes="0x01"
    MaxPktSize="1024"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="6"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISO"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISO"
        Direction="OUT"
        Address="0x06"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
</CONFIGURATION>

```

### 4.12.1 ToString()

public override string **ToString()**  
Member of [CyUSB.CyUSBConfig](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ToString returns an XML string that represents the USB descriptor for the device configuration.

#### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;

string text = myDev.USBConfigs[0].ToString();
```

Fills text with the following:

```
<CONFIGURATION>
  Configuration="0"
  ConfigurationValue="1"
  Attributes="0xA0"
  Interfaces="1"
  DescriptorType="2"
  DescriptorLength="9"
  TotalLength="135"
  MaxPower="50"
  <INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="0"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
      Type="BULK"
      Direction="IN"
      Address="0x82"
      Attributes="0x02"
      MaxPktSize="512"
      DescriptorType="5"
      DescriptorLength="7"
      Interval="0"
    </ENDPOINT>
  </INTERFACE>
  <INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="1"
    Class="0xFF"
    Subclass="0x00"
```

```

Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="BULK"
    Direction="OUT"
    Address="0x02"
    Attributes="0x02"
    MaxPktSize="512"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="0"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"

```



```
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="3072"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="4"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
  <ENDPOINT>
    Type="ISOC"
    Direction="OUT"
    Address="0x02"
    Attributes="0x01"
    MaxPktSize="3072"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
  </ENDPOINT>
</INTERFACE>
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="5"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
  <ENDPOINT>
    Type="ISOC"
    Direction="IN"
    Address="0x82"
    Attributes="0x01"
    MaxPktSize="1024"
    DescriptorType="5"
    DescriptorLength="7"
```

```
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="6"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x06"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
</CONFIGURATION>
```

### 4.12.2 AltInterfaces

```
public byte AltInterfaces { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

#### Description

AltInterfaces returns the total number of interfaces exposed by the configuration (including the default interface). This value is the number of interface descriptors contained in the current configuration descriptor.

### 4.12.3 bConfigurationValue

```
public byte bConfigurationValue { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

#### Description

bConfigurationValue contains value of the *bConfigurationValue* field from the selected configuration descriptor.

#### 4.12.4 bDescriptorType

```
public byte bDescriptorType { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

##### Description

bDescriptorType contains value of the **bDescriptorType** field from the selected configuration descriptor.

#### 4.12.5 bLength

```
public byte bLength { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

##### Description

bLength contains value of the **bLength** field from the selected configuration descriptor.

#### 4.12.6 bmAttributes

```
public byte bmAttributes { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

##### Description

bmAttributes contains value of the **bmAttributes** field from the selected configuration descriptor.

### 4.12.7 bNumInterfaces

```
public byte bNumInterfaces { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

#### Description

bNumInterfaces contains value of the **bNumInterfaces** field from the selected configuration descriptor.



#### 4.12.8 iConfiguration

```
public byte iConfiguration { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

##### Description

iConfiguration contains value of the **iConfiguration** field from the selected configuration descriptor.

### 4.12.9 MaxPower

```
public byte MaxPower { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

#### Description

MaxPower contains a value representing 1/2 the maximum power drawn by the device, expressed in mA. This value corresponds to the *bMaxPower* field of the configuration descriptor.

## 4.12.10 Tree

```
public System.Windows.Forms.TreeNode Tree { get;  
}
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

### Description

The Tree property returns a Windows.Forms.TreeNode.

The *Text* property of the TreeNode will be either "*Primary Configuration*" or "*Secondary Configuration*" as the CyUSBDevice class only accommodates up to 2 configurations per device.

The children of the returned node is comprised of a node representing the [Control Endpoint](#) for the configuration, followed by the trees representing the [CyUSBInterfaces](#) of the configuration.

The *Tag* property of the returned TreeNode contains a reference to the CyUSBConfig object (*this*).

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyHidDevice myDev = usbDevices[0] as CyUSBDevice;  
  
TreeNode cfgTree = myDev.USBCfgs[0].Tree;
```

### 4.12.11 wTotalLength

```
public ushort wTotalLength { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

#### Description

wTotalLength contains value of the *wTotalLength* field from the selected configuration descriptor.

## 4.12.12 Interfaces

public [CyAPI.CyUSBInterface\[\]](#) **Interfaces**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBConfig](#)

### Description

Interfaces is an array of CyUSBInterface objects. One CyUSBInterface object exists in the Interfaces array for each alternate interface exposed by the configuration (including alt setting 0).

The [AltInterfaces](#) member tells how many valid entries are held in Interfaces.

Use [CyUSBDevice.AltIntfc](#) property to evaluate or change the Alt Interface setting of the device.

The following example code shows how you might use the Interfaces array in an application.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;
```

```
for (byte i=0; i < MyDevice.AltIntfcCount; i++)
    DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].ToString();
```

Fills DescText.Text with the following:

```
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="0"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="1"
```

```

Class="0xFF"
Subclass="0x00"
Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="BULK"
    Direction="OUT"
    Address="0x02"
    Attributes="0x02"
    MaxPktSize="512"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="0"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"

```

```
Subclass="0x00"
Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="ISOC"
    Direction="IN"
    Address="0x82"
    Attributes="0x01"
    MaxPktSize="3072"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="4"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="5"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
```

```
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="6"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x06"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
```



## 4.13 CyUSBDevice

public class **CyUSBDevice** : [CyUSB.USBDevice](#)

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The CyUSBDevice class represents a USB device attached to the cyusb3.sys device driver.

A list of CyUSBDevice objects can be generated by passing [DEVICES\\_CYUSB](#) mask to the [USBDeviceList](#) constructor.

Once you obtain a [CyUSBDevice](#) object, you can communicate with the device via the objects various endpoint ([ControlEndPoint](#), [BulkInEndPoint](#), [BulkOutEndPoint](#), etc.) members.

Because CyUSBDevice is a descendant of [USBDevice](#), it inherits all the members of [USBDevice](#).

This class support both USB3.0 and USB2.0 device specific functionality.

### C# Example

```
CyControlEndPoint    CtrlEndPoint    = null;

USBDeviceList        usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice          MyDevice        = usbDevices[0x04B4,0x1003] as CyUSBDevice ;

if (MyDevice != null)
    CtrlEndPoint = MyDevice.ControlEndPoint;

if (CtrlEndPoint != null)
{
    CtrlEndPoint.Target      = CyConst.TGT_DEVICE;
    CtrlEndPoint.ReqType     = CyConst.REQ_STD;
    CtrlEndPoint.Direction  = CyConst.DIR_FROM_DEVICE;
    CtrlEndPoint.ReqCode     = 0x06;           // Get Descriptor Standard Request
    CtrlEndPoint.Value       = 0x200;         // Configuration Descriptor
    CtrlEndPoint.Index       = 0;

    int len                  = 256;
    byte[] buf               = new byte[len];

    CtrlEndPoint.XferData(ref buf, ref len);
}
```

### 4.13.1 EndPointOf( )

```
public CyUSB.CyUSBEndPoint EndPointOf ( byte  
addr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

Returns the CyUSBEndPoint object whose [Address](#) property is equal to *addr*.

Returns null if no endpoint with Address = *addr* is found.

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
CyUSBEndPoint ept = MyDevice.EndPointOf(0x82);  
  
if ((ept != null) && (ept.Attributes == 2))  
    Console.WriteLine("Found Bulk IN endpoint with address 0x82");
```

### 4.13.2 GetConfigDescriptor( )

```
public void GetConfigDescriptor ( ref CyUSB.  
USB\_CONFIGURATION\_DESCRIPTOR descr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This function copies the device's configuration descriptor into *descr*.

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_CONFIGURATION_DESCRIPTOR descriptor = new USB_CONFIGURATION_DESCRIPTOR();  
  
MyDevice.GetConfigDescriptor(ref descriptor);
```

### 4.13.3 GetDeviceDescriptor( )

```
public void GetDeviceDescriptor ( ref CyUSB.  
USB\_DEVICE\_DESCRIPTOR descr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This function copies the device's device descriptor into *descr*.

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_DEVICE_DESCRIPTOR descriptor = new USB_DEVICE_DESCRIPTOR();  
  
MyDevice.GetDeviceDescriptor(ref descriptor);
```

#### 4.13.4 GetIntfcDescriptor( )

```
public void GetIntfcDescriptor ( ref CyUSB.  
USB\_INTERFACE\_DESCRIPTOR descr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

##### Description

This function copies the device's interface descriptor into *descr*.

##### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_INTERFACE_DESCRIPTOR descriptor = new USB_INTERFACE_DESCRIPTOR();  
  
MyDevice.GetIntfcDescriptor(ref descriptor);
```

### 4.13.5 GetBosDescriptor()

```
public bool GetBosDescriptor ( ref CyUSB.  
USB\_BOS\_DESCRIPTOR descr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This function copies the device's Binary device object store descriptor into *descr*. This function will return BOS descriptor only for usb3.0 device.

#### Return Value

True Operation successful.

False Operation failed ( Device is not a usb3.0)

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_BOS_DESCRIPTOR descriptor = new USB_BOS_DESCRIPTOR();  
  
if(MyDevice.GetBosDescriptor(ref descriptor) ==false)  
    Console.WriteLine("Not a usb3.0 device");
```

#### 4.13.6 GetBosUSB20DeviceExtensionDescriptor()

```
public bool  
GetBosUSB20DeviceExtensionDescriptor ( ref  
CyUSB.USB\_BOS\_USB20\_DEVICE\_EXTENSION descr  
)  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

##### Description

This function copies the device's USB2.0 device extension descriptor into *descr*. This function will return USB2.0 Device extension descriptor only for usb3.0 device.

##### Return Value

True Operation successful.  
False Operation failed ( Device is not a usb3.0)

##### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_BOS_USB20_DEVICE_EXTENSION descriptor = new USB_BOS_USB20_DEVICE_EXTENSION();  
  
if(MyDevice.GetBosUSB20DeviceExtensionDescriptor(ref descriptor) ==false)  
Console.WriteLine("Not a usb3.0 device");
```

#### 4.13.7 GetBosSSCapabilityDescriptor()

```
public bool GetBosSSCapabilityDescriptor ( ref  
CyUSB.USB\_BOS\_SS\_DEVICE\_CAPABILITY descr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

##### Description

This function copies the device's super speed capability descriptor into *descr*. This function will return super speed capability descriptor only for usb3.0 device.

##### Return Value

True Operation successful.  
False Operation failed ( Device is not a usb3.0)

##### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_BOS_SS_DEVICE_CAPABILITY descriptor = new USB_BOS_SS_DEVICE_CAPABILITY();  
  
if(MyDevice.GetBosSSCapabilityDescriptor(ref descriptor) ==false)  
Console.WriteLine("Not a usb3.0 device");
```



### 4.13.8 GetBosContainerIDDescriptor()

```
public bool GetBosContainerIDDescriptor ( ref  
CyUSB.USB\_BOS\_CONTAINER\_ID descr )  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This function copies the device's Container ID descriptor into *descr*. This function will return Container ID only for usb3.0 device.

#### Return Value

True Operation successful.

False Operation failed ( Device is not a usb3.0)

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
USB_BOS_CONTAINER_ID descriptor = new USB_BOS_CONTAINER_ID();  
  
if(MyDevice.GetBosContainerIDDescriptor(ref descriptor) ==false)  
Console.WriteLine("Not a usb3.0 device");
```

### 4.13.9 ReConnect()

public bool **ReConnect** ( )  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ReConnect causes the device to be logically disconnected from the USB bus and re-enumerated.

#### 4.13.10 Reset( )

public bool **Reset** ( )  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

##### Description

Reset causes the USB device to be reset to its initial power-on configuration.

### 4.13.11 ToString( )

public override string **ToString( )**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ToString returns an XML string that represents the USB descriptor for the device.

#### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;

string devText = myDev.ToString();
```

Fills devText with the following:

```
<DEVICE>
  FriendlyName="CYStream DevKit Device"
  Manufacturer="Cypress"
  Product="CY-Stream"
  SerialNumber=""
  Configurations="1"
  MaxPacketSize="64"
  VendorID="0x04B4"
  ProductID="0x1003"
  Class="0x00"
  SubClass="0x00"
  Protocol="0x00"
  BcdDevice="0x0000"
  BcdUSB="0x0200"
<CONFIGURATION>
  Configuration="0"
  ConfigurationValue="1"
  Attributes="0xA0"
  Interfaces="1"
  DescriptorType="2"
  DescriptorLength="9"
  TotalLength="135"
  MaxPower="50"
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="0"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
<ENDPOINT>
  Type="BULK"
```

```
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="1"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x02"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
```

```

        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="4"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"

```

```
AltSetting="5"
Class="0xFF"
Subclass="0x00"
Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="6"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="2"
  DescriptorType="4"
  DescriptorLength="9"
  <ENDPOINT>
    Type="ISOC"
    Direction="IN"
    Address="0x82"
    Attributes="0x01"
    MaxPktSize="1024"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
  </ENDPOINT>
  <ENDPOINT>
    Type="ISOC"
    Direction="OUT"
    Address="0x06"
    Attributes="0x01"
    MaxPktSize="1024"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
  </ENDPOINT>
</INTERFACE>
</CONFIGURATION>
</DEVICE>
```

### 4.13.12 UsbdStatusString( )

public string **UsbdStatusString** ( uint *stat* )  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

The UsbdStatusString method returns a string that represents the UsbdStatus error code contained in *stat*.

The *stat* parameter should be the [UsbdStatus](#) member of a [CyUSBEndPoint](#) object.

The format of the returned string is:

"[state=SSSSSS status=TTTTTTTT]"

where SSSSSS can be "SUCCESS", "PENDING", "STALLED", or "ERROR".

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

string status;

if (MyDevice.BulkInEndPt != null)
{
    int len = 512;
    byte [] buf = new byte[len];

    MyDevice.BulkInEndPt.XferData(ref buf, ref len);

    status = CyUSBDevice.UsbdStatusString(MyDevice.BulkInEndPt.UsbdStatus);
}
```



### 4.13.13 AltIntfc

public byte **AltIntfc** { set; get; }  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

This property is used to get or set the alternate interface setting for the device.

Both the assignment and evaluation of AltIntfc result in communication with the device. Evaluation of AltIntfc (the get operation) queries the device to obtain its current Alt Setting. Assignment of a new value to AltIntfc sets the device's alternate interface setting to the new value if the new value is legitimate.

#### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.AltIntfc < 2)           // Queries the device
    MyDevice.AltIntfc = 2;           // Sets new value in device
```

#### 4.13.14 AltIntfcCount

public byte **AltIntfcCount** { get; }  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

##### Description

This property reports the number of alternate interfaces exposed by the device.

The primary interface (AltSetting == 0) is counted as an alternate interface.

An AltIntfcCount of 3 means that there are 3 alternate interfaces, including the primary interface. Legitimate [AltIntfc](#) values would then be 0, 1 and 2.

##### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
    DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].ToString();
```

Fills DescText.Text with the following:

```
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="0"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
<ENDPOINT>
    Type="BULK"
    Direction="IN"
    Address="0x82"
    Attributes="0x02"
    MaxPktSize="512"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="0"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="1"
    Class="0xFF"
```

```
Subclass="0x00"
Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="BULK"
    Direction="OUT"
    Address="0x02"
    Attributes="0x02"
    MaxPktSize="512"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="0"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"
    Subclass="0x00"
```

```

Protocol="0"
Endpoints="1"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="ISOC"
    Direction="IN"
    Address="0x82"
    Attributes="0x01"
    MaxPktSize="3072"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="4"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="5"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"

```

```
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="6"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x06"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
```

### 4.13.15 bHighSpeed

```
public bool bHighSpeed { get; }
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property evaluates to **true** if the USB device is a High Speed device.

#### C# Example

```
USBDeviceList Devices      = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
bool blsFast = MyDevice.bHighSpeed;
```

### 4.13.16 bSuperSpeed

```
public bool bSuperSpeed { get; }
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property evaluates to **true** if the USB device is a Super Speed device.

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;  
  
bool blsFast = MyDevice.bSuperSpeed;
```

### 4.13.17 BcdDevice

```
public ushort BcdDevice { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the value of the *bcdDevice* member from the device's USB descriptor structure.



### 4.13.18 Config

```
public byte Config { set; get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property is used to get or set the configuration index for the device.

Most devices only expose a single configuration at one time. So, **zero** is usually the only legitimate value for this property.

### 4.13.19 ConfigAttrib

```
public byte ConfigAttrib { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the value of the *bmAttributes* field from the device's current configuration descriptor.

### 4.13.20 ConfigCount

```
public byte ConfigCount { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the number of configurations reported by the device in the *bNumConfigurations* field of its device descriptor.

### 4.13.21 ConfigValue

```
public byte ConfigValue { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the value of the *bConfigurationValue* field from the device's current configuration descriptor.

### 4.13.22 DeviceHandle

```
public System.IntPtr DeviceHandle { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the object's open handle to the cyusb3.sys driver.

### 4.13.23 DriverVersion

```
public uint DriverVersion { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

DriverVersion returns 4 bytes representing the version of the driver that is attached to the device.

## 4.13.24 EndPointCount

public byte **EndPointCount** { get; }  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

### Description

This property returns the number of [CyUSBEndPoints](#) objects in the active Alternate Interface. This number will change depending on the number of endpoints for the currently selected [AltIntfc](#) setting.

The default Control endpoint (endpoint 0) is included in the count.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
    for (int e = 1; e < MyDevice.EndPointCount; e++)
        DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].Endpoints[e].ToString(); // DescText is System.
                                         Windows.Forms.TextBox
```

Fills DescText.Text with the following:

```
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="OUT"
  Address="0x02"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
```

```
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
```



```
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
  Address="0x06"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
```

### 4.13.25 IntfcClass

```
public byte IntfcClass { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)**Description**

This property returns the *bInterfaceClass* field from the currently selected interface's interface descriptor.

### 4.13.26 IntfcProtocol

```
public byte IntfcProtocol { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the *bInterfaceProtocol* field from the currently selected interface's interface descriptor.

### 4.13.27 IntfcSubClass

```
public byte IntfcSubClass { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the *bInterfaceSubClass* field from the currently selected interface's interface descriptor.

### 4.13.28 MaxPacketSize

```
public byte MaxPacketSize { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the value of the *bMaxPacketSize0* field from the open device's Device Descriptor structure.

### 4.13.29 MaxPower

```
public byte MaxPower { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

MaxPower returns a value representing 1/2 the maximum power drawn by the device, expressed in mA. This value corresponds to the *bMaxPower* field of the device's configuration descriptor.

### 4.13.30 StrLangID

```
public ushort StrLangID { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the value of *bString* field from the open device's first String Descriptor.

This value indicates the language of the other string descriptors.

If multiple languages are supported in the string descriptors and English is one of the supported languages, StrLangID is set to the value for English (0x0409).

### 4.13.31 Tree

```
public override System.Windows.Forms.TreeNode  
Tree { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBDevice](#)

#### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string returned by the [FriendlyName](#) property.

The children of the node are comprised of the trees representing the [USB configurations](#) of the device.

The Tag property of the returned TreeNode contains a reference to the CyUSBDevice object (*this*).

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```



### 4.13.32 USBDIVersion

```
public uint USBDIVersion { get; }  
Member of CyUSB.CyUSBDevice
```

[Top](#) [Previous](#) [Next](#)**Description**

This property returns the version of the USB Host Controller Driver in BCD format.

### 4.13.3 BulkInEndPt

public [CyUSB.CyBulkEndPoint](#) **BulkInEndPt**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

BulkInEndPt is a [CyBulkEndPoint](#) object representing the first BULK IN endpoint enumerated for the selected interface.

The selected interface might expose additional BULK IN endpoints. To discern this, one would need to traverse the [Endpoints](#) array, checking the [Attributes](#) and [Address](#) members of each [CyUSBEndPoint](#) object.

If no BULK IN endpoints were enumerated by the device, BulkInEndPt will be set to null.

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.BulkInEndPt != null)
{
    int len = 512;
    byte [] buf = new byte[len];

    MyDevice.BulkInEndPt.XferData(ref buf, ref len);
}
```

### 4.13.34 BulkOutEndPt

public [CyUSB.CyBulkEndPoint](#) **BulkOutEndPt**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

BulkOutEndPt is a [CyBulkEndPoint](#) object representing the first BULK OUT endpoint enumerated for the selected interface.

The selected interface might expose additional BULK OUT endpoints. To discern this, one would need to traverse the [EndPoints](#) array, checking the [Attributes](#) and [Address](#) members of each [CyUSBEndPoint](#) object.

If no BULK OUT endpoints were enumerated by the device, BulkOutEndPt will be set to null.

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.BulkOutEndPt != null)
{
    int len = 512;
    byte [] buf = new byte[len];

    MyDevice.BulkOutEndPt.XferData(ref buf, ref len);
}
```

### 4.13.35 ControlEndPt

public [CyUSB.CyControlEndPoint](#) **ControlEndPt**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ControlEndPt is a [CyControlEndPoint](#) object representing the primary Control endpoint of the device, endpoint 0.

ControlEndPt is a copy of [EndPoints](#)[0].

Before calling the [XferData](#) method for ControlEndPt, you should set the object's control properties.

#### C# Example

```
CyControlEndPoint CtrlEndPt    = null;

USBDeviceList usbDevices      = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice MyDevice          = usbDevices[0] as CyUSBDevice;

if (MyDevice != null)
    CtrlEndPt = MyDevice.ControlEndPt;

if (CtrlEndPt != null)
{
    CtrlEndPt.Target      = CyConst.TGT_DEVICE;
    CtrlEndPt.RegType     = CyConst.REQ_VENDOR;
    CtrlEndPt.RegCode     = 0xC2;
    CtrlEndPt.Value       = 2;
    CtrlEndPt.Index       = 0;

    int len               = 128;
    byte[] buf            = new byte[len];

    CtrlEndPt.Write(ref buf, ref len);

    bool success = (len == 128);
}
```

### 4.13.36 EndPoints

public [CyUSB.CyUSBEndPoint\[\]](#) **EndPoints**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

EndPoints is an array of references to [CyUSBEndPoint](#) objects.

The objects represent all the USB endpoints reported for the current [AltIntfc](#) of the device.

EndPoints[0] always contains a [CyControlEndPoint](#) object representing the primary Control Endpoint (endpoint 0) of the device.

Unused entries in EndPoints are set to null.

The [EndPointCount](#) property tells how many entries in EndPoints are valid.

EndPoints is re-populated each time a new [AltIntfc](#) value is assigned.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
{
    MyDevice.AltIntfc = i;

    for (int e = 1; e < MyDevice.EndPointCount; e++)
        DescText.Text += MyDevice.EndPoints[e].ToString(); // DescText is System.Windows.Forms.TextBox
}
```

Fills DescText.Text with the following:

```
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="OUT"
  Address="0x02"
  Attributes="0x02"
  MaxPktSize="512"
```

```
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
```

```
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
  Address="0x06"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
```

### 4.13.37 InterruptInEndPt

public [CyUSB.CyInterruptEndPoint](#)

[Top](#) [Previous](#) [Next](#)

**InterruptInEndPt**

Member of [CyUSB.CyUSBDevice](#)

#### Description

InterruptInEndPt is a [CyInterruptEndPoint](#) object representing the first INTERRUPT IN endpoint enumerated for the selected interface.

The selected interface might expose additional INTERRUPT IN endpoints. To discern this, one would need to traverse the [EndPoints](#) array, checking the [Attributes](#) and [Address](#) members of each [CyUSBEndPoint](#) object.

If no INTERRUPT IN endpoints were enumerated by the device, InterruptInEndPt will be set to null.

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.InterruptInEndPt != null)
{
    int len = 512;
    byte [] buf = new byte[len];

    MyDevice.InterruptInEndPt.XferData(ref buf, ref len);
}
```



### 4.13.38 InterruptOutEndPt

public [CyUSB.CyInterruptEndPoint](#)

[Top](#) [Previous](#) [Next](#)

#### **InterruptOutEndPt**

Member of [CyUSB.CyUSBDevice](#)

#### **Description**

InterruptOutEndPt is a [CyInterruptEndPoint](#) object representing the first INTERRUPT OUT endpoint enumerated for the selected interface.

The selected interface might expose additional INTERRUPT OUT endpoints. To discern this, one would need to traverse the [EndPoint](#)s array, checking the [Attributes](#) and [Address](#) members of each [CyUSBEndPoint](#) object.

If no INTERRUPT OUT endpoints were enumerated by the device, InterruptOutEndPt will be set to null.

#### **C# Example**

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.InterruptOutEndPt != null)
{
    int len = 512;
    byte [] buf = new byte[len];

    MyDevice.InterruptOutEndPt.XferData(ref buf, ref len);
}
```

### 4.13.39 IsocInEndPt

public [CyUSB.CyIsocEndPoint](#) **IsocInEndPt**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

IsocInEndPt is a [CyIsocEndPoint](#) object representing the first ISOC IN endpoint enumerated for the selected interface.

The selected interface might expose additional ISOC IN endpoints. To discern this, one would need to traverse the [EndPoints](#) array, checking the [Attributes](#) and [Address](#) members of each [CyUSBEndPoint](#) object.

If no ISOC IN endpoints were enumerated by the device, IsocInEndPt will be set to null.

#### C# Example

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.IsocInEndPt != null)
{
    int len = MyDevice.IsocInEndPt.MaxPktSize * 8;
    byte [] buf = new byte[len];

    MyDevice.IsocInEndPt.XferData(ref buf, ref len);
}
```

#### 4.13.40 IsocOutEndPt

public [CyUSB.CyIsocEndPoint](#) **IsocOutEndPt**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

##### Description

IsocOutEndPt is a [CyIsocEndPoint](#) object representing the first ISOC OUT endpoint enumerated for the selected interface.

The selected interface might expose additional ISOC OUT endpoints. To discern this, one would need to traverse the [EndPoints](#) array, checking the [Attributes](#) and [Address](#) members of each [CyUSBEndPoint](#) object.

If no ISOC OUT endpoints were enumerated by the device, IsocOutEndPt will be set to null.

##### C# Example

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

if (MyDevice.IsocOutEndPt != null)
{
    int len = MyDevice.IsocOutEndPt.MaxPktSize * 8;
    byte [] buf = new byte[len];

    MyDevice.IsocOutEndPt.XferData(ref buf, ref len);
}
```

#### 4.13.41 USBCfgs

public [CyUSB.CyUSBConfig](#)[] **USBCfgs**  
Member of [CyUSB.CyUSBDevice](#)

[Top](#) [Previous](#) [Next](#)

##### Description

USBCfgs is an array of [CyUSBConfig](#) objects representing the configuration descriptors returned by the device. Then number of elements in the array is indicated by the [ConfigCount](#) property (usually 1).

## 4.14 CyUSBEndPoint

public abstract class **CyUSBEndPoint**  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The CyUSBEndPoint class is abstract. The class contains many members which are common to all its descendants. So, you will need to be familiar with most of the members of CyUSBEndPoint.

Note that no public constructors for this class (or its descendants) is exposed. This is because endpoint objects are automatically instantiated for you (as part of a [USBDevice](#)) when you create a [USBDeviceList](#) object.

### 4.14.1 Abort( )

public bool **Abort** ( )  
Member of [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

#### Description

The Abort method sends an IOCTL\_ADAPT\_ABORT\_PIPE command to the the USB device driver, with the endpoint address as a parameter. This causes an abort of pending IO transactions on the endpoint.

#### C# Example

```
unsafe void function()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;

    byte[] overLap = new byte[CyConst.OverlapSignalAllocSize];

    fixed (byte* tmp0 = overLap)
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        if (!MyDevice.BulkInEndPt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            MyDevice.BulkInEndPt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);
        }
    }
}
```

#### 4.14.2 BeginDataXfer( )

```
unsafe public virtual bool BeginDataXfer ( ref byte
[] singleXfer, ref byte[] buffer, ref int len, ref byte
[] ov )
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

BeginDataXfer is an advanced method for performing asynchronous IO. This method sets-up all the parameters for a data transfer, initiates the transfer, and immediately returns, not waiting for the transfer to complete.

You will usually want to use the synchronous [XferData](#) method rather than the asynchronous BeginDataXfer/WaitForXfer/FinishDataXfer approach.

Again, the use of BeginDataXfer, [WaitForXfer](#), and [FinishDataXfer](#) is the difficult way to transfer data to and from a USB device. This approach should only be used if it is imperative that you squeeze every last bit of throughput from the USB.

If user set the XMODE to BUFFERED mode for particular endpoint then user need to allocate singleXfer (the command buffer) with size of SINGLE\_XFER\_LEN and data buffer length.

This buffer will be passed to the singleXfer the first parameter of BeginDataXfer. This is the requirement specific to the BUFFERED mode only. The below sample example shows the usage of it.

The code below utilizes the asynchronous methods to queue multiple transfers so as to keep the USB bandwidth fully utilized.

##### Advanced C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
public unsafe void ListenThread()
{
    if (MyDevice == null) return;

    CyBulkEndPoint InEndpt = MyDevice.BulkInEndPt;

    byte i = 0;

    int BufSz = InEndpt.MaxPktSize * Convert.ToUInt16(PpxBox.Text);
    int QueueSz = Convert.ToUInt16(QueueBox.Text);

    InEndpt.XferSize = BufSz;

    // Setup the queue buffers
    byte[][] cmdBufs = new byte[QueueSz][];
    byte[][] xferBufs = new byte[QueueSz][];
    byte[][] ovLaps = new byte[QueueSz][];

    for (i=0; i<QueueSz; i++)
    {
        cmdBufs[i] = new byte[CyConst.SINGLE_XFER_LEN+((InEndpt.XferMode == XMODE.BUFFERED) ?
        BufSz : 0)];
        xferBufs[i] = new byte[BufSz];
    }
}
```

```

        ovLaps[i] = new byte[CyConst.OverlapSignalAllocSize];

        fixed( byte *tmp0 = ovLaps[i])
        {
            OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
            ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
        }
    }

    // Pre-load the queue with requests
    int len = BufSz;

    for (i=0; i<QueueSz; i++)
        InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

    i = 0;
    int Successes = 0;
    int Failures = 0;

    XferBytes = 0;
    t1 = DateTime.Now;

    for (; StartBtn.Text.Equals("Stop");)
    {
        fixed( byte *tmp0 = ovLaps[i])
        {
            OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
            if (!InEndpt.WaitForXfer(ovLapStatus->hEvent, 500))
            {
                InEndpt.Abort();
                PInvoke.WaitForSingleObject(ovLapStatus->hEvent, CyConst.INFINITE);
            }
        }

        if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]))
        {
            XferBytes += len;
            Successes++;
        }
        else
            Failures++;

        // Re-submit this buffer into the queue
        len = BufSz;
        InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

        i++;

        if (i == QueueSz)
        {
            i = 0;
            t2 = DateTime.Now;

            elapsed = t2-t1;
            xferRate = (long)(XferBytes / elapsed.TotalMilliseconds);
            xferRate = xferRate / (int)100 * (int)100;

            if (xferRate > ProgressBar.Maximum)

```



```
        ProgressBar.Maximum = (int)(xferRate * 1.25);

        ProgressBar.Value = (int) xferRate;
        ThroughputLabel.Text = ProgressBar.Value.ToString();

        SuccessBox.Text = Successes.ToString();
        FailuresBox.Text = Failures.ToString();

        Thread.Sleep(0);
    }
}
}
```

### 4.14.3 FinishDataXfer( )

```
unsafe public virtual bool FinishDataXfer ( ref byte
[] singleXfer, ref byte[] buffer, ref int len, ref byte
[] ov )
Member of CyAPI.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

#### Description

FinishDataXfer is an advanced method for performing asynchronous IO. This method completes the data transfer that was initiated by the [BeginDataXfer](#) method.

You will usually want to use the synchronous [XferData](#) method rather than the asynchronous BeginDataXfer/WaitForXfer/FinishDataXfer approach.

Again, the use of [BeginDataXfer](#), [WaitForXfer](#), and FinishDataXfer is the difficult way to transfer data to and from a USB device. This approach should only be used if it is imperative that you squeeze every last bit of throughput from the USB.

The code, below, utilizes the asynchronous methods to queue multiple transfers so as to keep the USB bandwidth fully utilized.

#### Advanced C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
public unsafe void ListenThread()
{
    if (MyDevice == null) return;

    CyBulkEndPoint InEndpt = MyDevice.BulkInEndPt;

    byte i = 0;

    int BufSz = InEndpt.MaxPktSize * Convert.ToInt16(PpxBox.Text);
    int QueueSz = Convert.ToInt16(QueueBox.Text);

    InEndpt.XferSize = BufSz;

    // Setup the queue buffers
    byte[][] cmdBufs = new byte[QueueSz][];
    byte[][] xferBufs = new byte[QueueSz][];
    byte[][] ovLaps = new byte[QueueSz][];

    for (i=0; i<QueueSz; i++)
    {
        cmdBufs[i] = new byte[CyConst.SINGLE_XFER_LEN+ ((InEndpt.XferMode == XMODE.BUFFERED) ?
BufSz : 0)];
        xferBufs[i] = new byte[BufSz];
        ovLaps[i] = new byte[CyConst.OverlapSignalAllocSize];

        fixed( byte *tmp0 = ovLaps[i])
        {
            OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
            ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
        }
    }
}
```

```

// Pre-load the queue with requests
int len = BufSz;

for (i=0; i<QueueSz; i++)
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

i = 0;
int Successes = 0;
int Failures = 0;

XferBytes = 0;
t1 = DateTime.Now;

for (; StartBtn.Text.Equals("Stop");)
{
    fixed( byte *tmp0 = ovLaps[i])
    {
        OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
        if (!InEndpt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            InEndpt.Abort();
            Pnvoke.WaitForSingleObject(ovLapStatus->hEvent, CyConst.INFINITE);
        }
    }

    if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]))
    {
        XferBytes += len;
        Successes++;
    }
    else
        Failures++;

    // Re-submit this buffer into the queue
    len = BufSz;
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

    i++;

    if (i == QueueSz)
    {
        i = 0;
        t2 = DateTime.Now;

        elapsed = t2-t1;
        xferRate = (long)(XferBytes / elapsed.TotalMilliseconds) ;
        xferRate = xferRate / (int)100 * (int)100;

        if (xferRate > ProgressBar.Maximum)
            ProgressBar.Maximum = (int)(xferRate * 1.25);

        ProgressBar.Value = (int) xferRate;
        ThroughputLabel.Text = ProgressBar.Value.ToString();

        SuccessBox.Text = Successes.ToString();
        FailuresBox.Text = Failures.ToString();

        Thread.Sleep(0);
    }
}

```

```
        }  
    }  
}
```

#### 4.14.4 Reset()

```
public bool Reset ( )  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

The Reset method resets the endpoint, clearing any error or stall conditions on that endpoint.

Pending data transfers are not cancelled by the Reset method.

Call [Abort](#) for the endpoint in order force completion of any transfers in-process.

#### 4.14.5 ToString()

public override string **ToString()**  
Member of [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

##### Description

ToString returns an XML string that describes the endpoint descriptor.

##### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
    for (int e = 1; e < MyDevice.EndPointCount; e++)
        DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].Endpoints[e].ToString();
```

Fills DescText.Text with the following:

```
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="OUT"
  Address="0x02"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
```

```
<ENDPOINT>
  Type="BULK"
  Direction="OUT"
  Address="0x06"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="3072"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
  Address="0x02"
  Attributes="0x01"
  MaxPktSize="3072"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
```

```
Address="0x06"  
Attributes="0x01"  
MaxPktSize="1024"  
DescriptorType="5"  
DescriptorLength="7"  
Interval="1"  
</ENDPOINT>
```



#### 4.14.6 WaitForXfer( )

```
public bool WaitForXfer ( uint overlapEvent , uint
                        tOut )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBEndPoint](#)

##### Description

WaitForXfer is an advanced method for performing asynchronous IO. This method waits *tOut* milliseconds for the transfer associated with *overlapEvent* to complete.

You will usually want to use the synchronous [XferData](#) method rather than the asynchronous BeginDataXfer/WaitForXfer/FinishDataXfer approach.

Again, the use of [BeginDataXfer](#), WaitForXfer, and [FinishDataXfer](#) is the difficult way to transfer data to and from a USB device. This approach should only be used if it is imperative that you squeeze every last bit of throughput from the USB.

The code, below, utilizes the asynchronous methods to queue multiple transfers so as to keep the USB bandwidth fully utilized.

##### Advanced C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
public unsafe void ListenThread()
{
    if (MyDevice == null) return;

    CyBulkEndPoint InEndpt = MyDevice.BulkInEndPt;

    byte i = 0;

    int BufSz = InEndpt.MaxPktSize * Convert.ToInt16(PpxBox.Text);
    int QueueSz = Convert.ToInt16(QueueBox.Text);

    InEndpt.XferSize = BufSz;

    // Setup the queue buffers
    byte[][] cmdBufs    = new byte[QueueSz][];
    byte[][] xferBufs   = new byte[QueueSz][];
    byte[][] ovLaps     = new byte[QueueSz][];

    for (i=0; i<QueueSz; i++)
    {
        cmdBufs[i]      = new byte[CyConst.SINGLE_XFER_LEN+ ((InEndpt.XferMode == XMODE.BUFFERED) ?
        BufSz : 0)];
        xferBufs[i]     = new byte[BufSz];
        ovLaps[i]       = new byte[CyConst.OverlapSignalAllocSize];

        fixed( byte *tmp0 = ovLaps[i])
        {
            OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
            ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);
        }
    }
}
```

```

// Pre-load the queue with requests
int len = BufSz;

for (i=0; i<QueueSz; i++)
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

i = 0;
int Successes = 0;
int Failures = 0;

XferBytes = 0;
t1 = DateTime.Now;

for (; StartBtn.Text.Equals("Stop");)
{
    fixed( byte *tmp0 = ovLaps[i])
    {
        OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;
        if (!InEndpt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            InEndpt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, CyConst.INFINITE);
        }
    }

    if (InEndpt.FinishDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]))
    {
        XferBytes += len;
        Successes++;
    }
    else
        Failures++;

    // Re-submit this buffer into the queue
    len = BufSz;
    InEndpt.BeginDataXfer(ref cmdBufs[i], ref xferBufs[i], ref len, ref ovLaps[i]);

    i++;

    if (i == QueueSz)
    {
        i = 0;
        t2 = DateTime.Now;

        elapsed = t2-t1;
        xferRate = (long)(XferBytes / elapsed.TotalMilliseconds) ;
        xferRate = xferRate / (int)100 * (int)100;

        if (xferRate > ProgressBar.Maximum)
            ProgressBar.Maximum = (int)(xferRate * 1.25);

        ProgressBar.Value = (int) xferRate;
        ThroughputLabel.Text = ProgressBar.Value.ToString();

        SuccessBox.Text = Successes.ToString();
        FailuresBox.Text = Failures.ToString();

        Thread.Sleep(0);
    }
}

```

```
}  
}
```

#### 4.14.7 XferData( )

```
unsafe public virtual bool XferData(ref byte[] buf,
ref int len)
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

The XferData method sends or receives *len* bytes of data from / into *buf*.

This is the primary IO method of the library for transferring data. It performs synchronous (i.e. blocking) IO operations and does not return until the transaction completes or the endpoint's [TimeOut](#) has elapsed. It call Abort() method internally if operation fail.

For all non-control endpoints, the direction of the transfer is implied by the endpoint itself. (Each such endpoint will either be an IN or an OUT endpoint.)

For control endpoints, the [Direction](#) must be specified, along with the other control-specific parameters.

Returns *true* if the transaction successfully completes before [TimeOut](#) has elapsed.

Note that the *len* parameter is a reference, meaning that the method can modify its value. The number of bytes actually transferred is passed back in *len*.

##### C# Example

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice      MyDevice      = usbDevices[0x04B4,0x1003] as CyUSBDevice;

if (MyDevice != null)
    if (MyDevice.BulkOutEndPt != null)
    {
        int len = 512;
        byte [] buf = new byte[len];
        MyDevice.BulkOutEndPt.XferData(ref buf, ref len);
    }
```

#### 4.14.8 Address

public byte **Address** { get; }  
Member of [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

##### Description

*Address* returns the value of the bEndpointAddress field of the endpoint descriptor returned by the device.

Addresses with the high-order bit set (0x8\_) are IN endpoints.

Addresses with the high-order bit cleared (0x0\_) are OUT endpoints.

The default control endpoint, [ControlEndPt](#), has Address = 0.

##### Example

```
// Find a second Bulk IN endpoint in the EndPoints[] array

CyBulkEndPoint BulkIn2 = null;

// Create a list of devices served by cyusb3.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
if (usbDevices.Count == 0) return;

// Just look at the first device in the list
CyUSBDevice dev = usbDevices[0] as CyUSBDevice;

int e = 0;

do {
    CyUSBEndPoint ept = dev.EndPoints[e];

    bool bIn = ((ept.Address & 0x80) > 0);
    bool bBulk = (ept.Attributes == 2);

    if (bBulk && bIn)
        BulkIn2 = (CyBulkEndPoint) ept;

    e++;
} while ( ( e < dev.EndPointCount ) && ( BulkIn2 == null ) );
```

### 4.14.9 Attributes

public byte **Attributes** { get; }  
Member of [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

#### Description

Attributes returns the value of the *bmAttributes* field of the endpoint's descriptor.

The Attributes member indicates the type of endpoint per the following list.

- 0: Control
- 1: Isochronous
- 2: Bulk
- 3: Interrupt

#### C# Example

```
// Find a second Bulk IN endpoint in the EndPoints[] array

CyBulkEndPoint BulkIn2 = null;

// Create a list of devices served by cyusb3.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
if (usbDevices.Count == 0) return;

// Just look at the first device in the list
CyUSBDevice dev = usbDevices[0] as CyUSBDevice;

int e = 0;

do {
    CyUSBEndPoint ept = dev.EndPoints[e];

    bool bIn = ((ept.Address & 0x80) > 0);
    bool bBulk = (ept.Attributes == 2);

    if (bBulk && bIn)
        BulkIn2 = (CyBulkEndPoint) ept;

    e++;
} while ( (e < dev.EndPointCount) && (BulkIn2 == null) );
```

#### 4.14.10 bln

```
public bool bln { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

bln indicates whether or not the endpoint is an IN endpoint.

IN endpoints transfer data from the USB device to the Host (PC).

Endpoint addresses with the high-order bit set (0x8\_) are IN endpoints. Endpoint addresses with the high-order bit cleared (0x0\_) are OUT endpoints.

bln is not valid for [CyControlEndPoint](#) objects.

##### Example

```
// Find a second Bulk IN endpoint in the EndPoints[] array  
  
CyBulkEndPoint BulkIn2 = null;  
  
// Create a list of devices served by cyusb3.sys  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Just look at the first device in the list  
CyUSBDevice dev = usbDevices[0] as CyUSBDevice;  
  
int e = 0;  
  
do {  
    CyUSBEndPoint ept = dev.EndPoints[e];  
  
    bool bBulk = (ept.Attributes == 2);  
  
    if (bBulk && ept.blb)  
        BulkIn2 = (CyBulkEndPoint) ept;  
  
    e++;  
  
} while ( ( e < dev.EndPointCount ) && ( BulkIn2 == null ) );
```

#### 4.14.11 BytesWritten

```
public uint BytesWritten { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

BytesWritten contains the number of data buffer bytes transferred to or from the endpoint in the most recent [XferData](#) or [FinishDataXfer](#) call.



#### 4.14.12 DscLen

```
public byte DscLen { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

DscLen contains the length of the endpoint descriptor as reported in the *bLength* field of the USB\_ENDPOINT\_DESCRIPTOR structure that was passed to the endpoint object's constructor. (Because the passed descriptor was an endpoint descriptor, this value should always be 0x07.)

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.14.13 DscType

```
public byte DscType { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

DscType contains the type of the endpoint descriptor as reported in the *bDescriptorType* field of the USB\_ENDPOINT\_DESCRIPTOR structure that was passed to the endpoint object's constructor. (Because the passed descriptor was an endpoint descriptor, this value should always be 0x05.)

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.14.14 hDevice

```
public System.IntPtr hDevice { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

hDevice contains a handle to the USB device driver, through which all the IO is carried-out.

The only reason to access this data member would be to call the device driver explicitly, bypassing the API library methods. *This is not recommended.*

You should never call the Windows CloseHandle(hDevice) directly as this happens automatically when a [CyUSBDevice](#) object is destroyed.

Note that an instance of [CyUSBDevice](#) will contain several [CyUSBEndPoint](#) objects. Each of those will have the same value for their hDevice member. This value will also match the [DeviceHandle](#) property of the [CyUSBDevice](#).

#### 4.14.15 Interval

```
public byte Interval { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

Interval contains the value reported in the *bInterval* field of the USB\_ENDPOINT\_DESCRIPTOR structure that was passed to the endpoint object's constructor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.14.16 MaxPktSize

```
public int MaxPktSize { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

MaxPktSize contains the value indicated by the *wMaxPacketSize* field of the USB\_ENDPOINT\_DESCRIPTOR structure that was passed to the endpoint object's constructor.

MaxPktSize is calculated by multiplying the low-order 11 bits of *wMaxPacketSize* by the value represented by 1 + the next 2 bits (bits 11 and 12) .

For USB3.0 device the MaxPktSize contains the value indicated by *wMaxPacketSize* field of the USB\_ENDPOINT\_DESCRIPTOR structure and multiply it with the SSMaxBurst field of Super speed companion descriptor.

##### Example

If wMaxPacketSize is 0x1400 (binary = 0001 0100 0000 0000)

MaxPktSize = [100 0000 0000 binary] \* [10 binary + 1] = 1024 \* 3 = 3072

For USB3.0 Device.

SSMaxBurst = 3

*wMaxPacketSize* = 1024

MaxPktSize = (*wMaxPacketSize* \* (SSMaxBurst+1))

#### 4.14.17 NtStatus

```
public uint NtStatus { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

NtStatus member contains the error code returned from the last call to the XferData or BeginDataXfer methods.

#### 4.14.18 TimeOut

public uint **TimeOut** { set; get; }  
Member of [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

##### Description

TimeOut limits the length of time that a [XferData](#) call will wait for the transfer to complete.

The units of TimeOut are milliseconds.

NOTE : For [CyControlEndPoint](#), the TimeOut is rounded down to the nearest 1000 ms, except for values between 0 and 1000 which are rounded up to 1000.

Set the TimeOut values to 0xFFFFFFFF(INFINITE), to wait for infinite time on the any transfers(bulk, Isochronous, Interrupt, and Control).

The TimeOut value 0 for bulk, interrupt, and isochronous transfers does not wait for read/write operation to complete, it will return immediately.

The TimeOut value 0 for control transfer is rounded up to 1000ms.

The default TimeOut for Bulk, Interrupt, Control, and Isochronous transfer is 10 seconds. User can override this value depending upon their application needs.

##### C# Example

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyUSBDevice      MyDevice      = usbDevices[0x04B4, 0x1003] as CyUSBDevice;

if (MyDevice != null)
    if (MyDevice.BulkOutEndPoint != null)
    {
        int len = 512;
        byte [] buf = new byte[len];

        MyDevice.BulkOutEndPoint.TimeOut = 5000; // 5 sec time out or set CyConst.INFINITE(0xFFFFFFFF) to wait forever.
        MyDevice.BulkOutEndPoint.XferData(ref buf, ref len);
    }
```

### 4.14.19 Tree

```
public System.Windows.Forms.TreeNode Tree { get;  
}
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBEndPoint](#)

#### Description

The `Tree` property returns a `Windows.Forms.TreeNode`.

The `Text` property of the `TreeNode` is the string describing the endpoint, with the endpoint address in parentheses, as shown here:

Bulk out endpoint (0x06)

The `TreeNode` of `CyUSBEndPoint` has no child nodes.

The `Tag` property of the returned `TreeNode` contains a reference to the `CyUSBEndpoint` object (*this*).

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```



#### 4.14.20 UsbdStatus

```
public uint UsbdStatus { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

UsbdStatus member contains an error code returned from the last call to the XferData or BeginDataXfer methods.

#### 4.14.21 XferMode

public byte **XferMode** { set; get; }  
Member of [CyUSB.CyUSBEndPoint](#)

[Top](#) [Previous](#) [Next](#)

##### Description

The [XferMode](#) property controls how data is passed to / from the cyusb3.sys driver.

The API would create a temporary buffer to pass to the driver, then copy the user's data to/from that buffer. This double buffering scheme incurred a performance penalty and was replaced by the more efficient direct transfer mode.

In direct transfer mode, the API passes the user's buffer to the driver and the driver accesses that buffer directly.

The default value of XferMode is [XMODE.DIRECT](#).

##### C# Example

```
USBDeviceList usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice  StreamDevice   = usbDevices["Cy Stream Device"] as CyUSBDevice;  
  
if (StreamDevice != null)  
    StreamDevice.BulkInEndPt.XferMode = XMODE.BUFFERED;    // Use old, slow, double buffering
```

#### 4.14.22 XferSize

```
public int XferSize { set; get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

This function is no longer supported. It is available to keep backward compatibility with legacy library and applications.

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

#### 4.14.23 SSDscLen

```
public int SSDscLen { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)**Description**

SSDscLen contains the length of the supperspeed endpoint companion descriptor as reported in the *bLength* field of the USB\_SUPERPEED\_ENDPOINT\_COMPANION\_DESCRIPTOR structure that was passed to the endpoint object's constructor. (Because the passed descriptor was an endpoint descriptor, this value should always be 0x06.)

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.14.24 SSDscType

```
public byte SSDscType { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

SSDscType contains the type of the superspeed endpoint companion descriptor as reported in the *bDescriptorType* field of the USB\_SUPERPEED\_ENDPOINT\_COMPANION\_DESCRIPTOR structure that was passed to the endpoint object's constructor. (Because the passed descriptor was an endpoint descriptor, this value should always be 0x30.)

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.14.25 SSMaxBurst

```
public byte SSMaxBurst { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

SSmaxBurst contains the value indicated by the bMaxBurst field of the USB\_SUPER SPEED\_ENDPOINT\_COMPANION\_DESCRIPTOR structure that was passed to the endpoint object's constructor.

The SSMaxBurst represent the maximum number of packets the endpoint can send or receive as part of a burst. Valid values are from 0 to 15. A value of 0 indicates that the endpoint can only burst one packet at a time and a value of 16 indicates that the endpoint can burst up to 16 packets at a time.

For endpoint of type control this shall be set to 0.

#### 4.14.26 SSBmAttribute

```
public byte SSBmAttribute { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

SSBmAttribute contains the value indicated by the bmAttributes field of the USB\_SUPERPEED\_ENDPOINT\_COMPANION\_DESCRIPTOR structure that was passed to the endpoint object's constructor.

SSBmAttribute represent different information based on the type of endpoint.

If this is a bulk endpoint:

Bits(4:0) - MaxStreams , The maximum number of stream this endpoint supports. Valid values are from 0 to 16, where a values of 0 indicates that the endpoint does not define streams. For the values 1 to 16 the number of streams supported equals  $\text{power}(2, \text{MaxStream})$ .

Bit (7:5) - Reserved. These bits are reserved and shall be set to zero.

If this a control or interrupt endpoint type:

(7:0) - Reserved. These bits are reserved and shall be set to zero.

if this is an isochronous endpoint:

Bits(1:0) Mult. A zero based value that determines the maximum number of packet within a service interval that this endpoint supports.

Maximum number of packets =  $\text{bMaxBurst} * (\text{Mult} + 1)$

The maximum value that can be set in this field is 2.

Bits(7:2) Reserved. These bits are reserved and shall be set to zero.

#### 4.14.27 SSBytePerInterval

```
public ushort SSBytePerInterval { get; }  
Member of CyUSB.CyUSBEndPoint
```

[Top](#) [Previous](#) [Next](#)

##### Description

SSBytePerInterval contains the value indicated by the bBytePerInterval field of the USB\_SUPER SPEED\_ENDPOINT\_COMPANION\_DESCRIPTOR structure that was passed to the endpoint object's constructor.

The total number of bytes this endpoint will transfer every service interval. This field is only valid for periodic endpoint.

For Isochronous endpoint, this value is used to reserve the bus time in the schedule, required for the frame data payloads per 125us. The pipe may, on an ongoing basis, actually use less bandwidth than that reserved. The device reports, if necessary, the actual bandwidth used via its normal, non-USB define mechanism.



## 4.15 CyUSBInterface

public class **CyUSBInterface**  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

CyUSBInterface represents a USB device interface. Such interfaces have one or more endpoints.

When a CyUSBDevice object is created, an instance of CyUSBConfig is constructed for each configuration reported by the device's device descriptor. (Normally, there is just one.)

In the process of construction, CyUSBConfig creates instances of CyUSBInterface for each interface exposed in the device's configuration descriptor. In turn, the CyUSBInterface class creates instances of [CyUSBEndPoint](#) for each endpoint descriptor contained in the interface descriptor. In this iterative fashion, the entire structure of Configs->Interfaces->EndPoints gets populated from a single construction of the CyUSBDevice class.

The below example code shows the use of the CyUSBInterface class in an application.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
{
    CyUSBInterface intfc = MyDevice.USBConfigs[0].Interfaces[i];
    DescText.Text += intfc.ToString();
}
```

Fills DescText.Text with the following:

```
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="0"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
```

```

        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="1"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x02"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="BULK"
        Direction="IN"
        Address="0x82"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="0"
    </ENDPOINT>
    <ENDPOINT>
        Type="BULK"
        Direction="OUT"
        Address="0x06"
        Attributes="0x02"
        MaxPktSize="512"
        DescriptorType="5"
        DescriptorLength="7"
    </ENDPOINT>

```

```
        Interval="0"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="4"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="5"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
```

```

DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
    Type="ISOC"
    Direction="IN"
    Address="0x82"
    Attributes="0x01"
    MaxPktSize="1024"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="1"
</ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="6"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x06"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>

```

### 4.15.1 ToString

public override string **ToString()**  
Member of [CyUSB.CyUSBInterface](#)

[Top](#) [Previous](#) [Next](#)

#### Description

ToString returns an XML string that represents a USB Interface descriptor.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
    DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].ToString();
```

Fills DescText.Text with the following:

```
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="0"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
  <ENDPOINT>
    Type="BULK"
    Direction="IN"
    Address="0x82"
    Attributes="0x02"
    MaxPktSize="512"
    DescriptorType="5"
    DescriptorLength="7"
    Interval="0"
  </ENDPOINT>
</INTERFACE>
<INTERFACE>
  Interface="0"
  InterfaceNumber="0"
  AltSetting="1"
  Class="0xFF"
  Subclass="0x00"
  Protocol="0"
  Endpoints="1"
  DescriptorType="4"
  DescriptorLength="9"
```

```

    <ENDPOINT>
      Type="BULK"
      Direction="OUT"
      Address="0x02"
      Attributes="0x02"
      MaxPktSize="512"
      DescriptorType="5"
      DescriptorLength="7"
      Interval="0"
    </ENDPOINT>
  </INTERFACE>
  <INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="2"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="2"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
      Type="BULK"
      Direction="IN"
      Address="0x82"
      Attributes="0x02"
      MaxPktSize="512"
      DescriptorType="5"
      DescriptorLength="7"
      Interval="0"
    </ENDPOINT>
    <ENDPOINT>
      Type="BULK"
      Direction="OUT"
      Address="0x06"
      Attributes="0x02"
      MaxPktSize="512"
      DescriptorType="5"
      DescriptorLength="7"
      Interval="0"
    </ENDPOINT>
  </INTERFACE>
  <INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="3"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>

```

```
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="4"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="OUT"
        Address="0x02"
        Attributes="0x01"
        MaxPktSize="3072"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
    Interface="0"
    InterfaceNumber="0"
    AltSetting="5"
    Class="0xFF"
    Subclass="0x00"
    Protocol="0"
    Endpoints="1"
    DescriptorType="4"
    DescriptorLength="9"
    <ENDPOINT>
        Type="ISOC"
        Direction="IN"
        Address="0x82"
        Attributes="0x01"
        MaxPktSize="1024"
        DescriptorType="5"
        DescriptorLength="7"
        Interval="1"
    </ENDPOINT>
</INTERFACE>
<INTERFACE>
```

```
Interface="0"
InterfaceNumber="0"
AltSetting="6"
Class="0xFF"
Subclass="0x00"
Protocol="0"
Endpoints="2"
DescriptorType="4"
DescriptorLength="9"
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
  Address="0x06"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
</INTERFACE>
```



### 4.15.2 bAlternateSetting

```
public byte bAlternateSetting { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the *bAlternateSetting* field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

### 4.15.3 bAltSettings

```
public byte bAltSettings { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the number of valid alternate interface settings exposed by this interface.

For an interface that exposes a primary interface and two alternate interfaces, this value would be 3.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

See [CyUSBDevice.AltIntfcCount](#)

#### 4.15.4 bDescriptorType

```
public byte bDescriptorType { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

##### Description

This property reports the *bDescriptorType* field of the USB\_INTERFACE\_DESCRIPTOR structure that was passed to the interface object's constructor. (Because the passed descriptor was an interface descriptor, this value should always be 0x04.)

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

### 4.15.5 **bInterfaceClass**

```
public byte bInterfaceClass { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

#### **Description**

This property reports the *bInterfaceClass* field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.15.6 bInterfaceNumber

```
public byte bInterfaceNumber { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

##### Description

This property reports the *bInterfaceNumber* field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

### 4.15.7 bInterfaceProtocol

```
public byte bInterfaceProtocol { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the *bInterfaceProtocol* field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.15.8 bInterfaceSubClass

```
public byte bInterfaceSubClass { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

##### Description

This property reports the *bInterfaceSubClass* field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

### 4.15.9 bLength

```
public byte bLength { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the *bLength* field from the currently selected interface's interface descriptor. It indicates the length of the interface descriptor. (Because the descriptor is an interface descriptor, this value should always be 0x09.)



#### 4.15.10 bNumEndpoints

```
public byte bNumEndpoints { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

##### Description

This property reports the *bNumEndpoints* field from the currently selected interface's interface descriptor. It indicates how many endpoint descriptors are returned for the selected interface.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

#### 4.15.11 *ilInterface*

```
public byte iInterface { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

##### Description

This property reports the *ilInterface* field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

## 4.15.12 Tree

```
public System.Windows.Forms.TreeNode Tree { get;  
}
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyUSBInterface](#)

### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string of the format "Alternate Interface n".

The children of the node are comprised of the trees representing the [endpoints](#) of the interface.

The Tag property of the returned TreeNode contains a reference to the CyUSBInterface object (*this*).

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void RefreshDeviceTree()  
{  
    DeviceTreeView.Nodes.Clear();  
    DescText.Text = "";  
  
    foreach (USBDevice dev in usbDevices)  
        DeviceTreeView.Nodes.Add(dev.Tree);  
}  
  
private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)  
{  
    TreeNode selNode = DeviceTreeView.SelectedNode;  
    DescText.Text = selNode.Tag.ToString();  
}
```

### 4.15.13 wTotalLength

```
public ushort wTotalLength { get; }  
Member of CyUSB.CyUSBInterface
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the **wTotalLength** field from the currently selected interface's interface descriptor.

This data member exists for completeness and debugging purposes. You should normally never need to access this data member.

## 4.15.14 EndPoints

public [CyUSB.CyUSBEndPoint\[\]](#) **EndPoints**  
Member of [CyUSB.CyUSBInterface](#)

[Top](#) [Previous](#) [Next](#)

### Description

This an array of CyUSBEndPoint objects that contain information about the endpoints of the interface.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList Devices    = new USBDeviceList(CyConst.DEVICES_CYUSB);

CyUSBDevice MyDevice = Devices[0] as CyUSBDevice;

for (byte i = 0; i < MyDevice.AltIntfcCount; i++)
    for (int e = 1; e < MyDevice.EndPointCount; e++)
        DescText.Text += MyDevice.USBCfgs[0].Interfaces[i].EndPoints[e].ToString();
```

Fills DescText.Text with the following:

```
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="OUT"
  Address="0x02"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="BULK"
  Direction="IN"
  Address="0x82"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
```

```
<ENDPOINT>
  Type="BULK"
  Direction="OUT"
  Address="0x06"
  Attributes="0x02"
  MaxPktSize="512"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="0"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="3072"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
  Address="0x02"
  Attributes="0x01"
  MaxPktSize="3072"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="IN"
  Address="0x82"
  Attributes="0x01"
  MaxPktSize="1024"
  DescriptorType="5"
  DescriptorLength="7"
  Interval="1"
</ENDPOINT>
<ENDPOINT>
  Type="ISOC"
  Direction="OUT"
```

```
Address="0x06"  
Attributes="0x01"  
MaxPktSize="1024"  
DescriptorType="5"  
DescriptorLength="7"  
Interval="1"  
</ENDPOINT>
```

## 4.16 CyUSBStorDevice

public class **CyUSBStorDevice** : [CyUSB.USBDevice](#)  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The CyUSBStorDevice class represents a USB Mass Storage Class device that is served by the Microsoft USB Mass Storage Class device driver, usbstor.sys.

Whereas earlier versions of the library only supported devices served by the CyUSB3.sys device driver, this class allows communication with mass storage class devices through the standard, Windows mass storage class device driver. This communication is accomplished via the SCSI Passthrough mechanism exposed by that driver.

The CyUSBStorDevice class gathers information about a mass storage device by searching the Windows registry for the device, based on the serial number reported in the device's [Path](#). So, only mass storage class devices that report a serial number string will work properly with the CyUSB library.

Because CyUSBStorDevice is a descendant of [USBDevice](#), it inherits all the members of [USBDevice](#).

### C# Example

```
// Create a list of devices served by the usbstor.sys driver
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC);
if (devList.Count == 0) return;

CyUSBStorDevice StorDevice = devList[0] as CyUSBStorDevice;

string SerNum = StorDevice.SerialNumber;
```



### 4.16.1 SendScsiCmd()

```
unsafe public bool SendScsiCmd(byte cmd, byte op
, byte lun, byte dirIn, int bank, int lba, int bytes,
byte[] data)
Member of CyUSB.CyUSBStorDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

The SendScsiCmd method uses the usbstor.sys driver's SCSI Pass-through facility to transfer data to and from the device. It uses the CDB10 structure for the Command Descriptor Block (CDB) passed to the device.

The **cmd** parameter contains a single-byte SCSI command code for the device.

The **op** parameter contains any argument needed for the **cmd** parameter. (For some SCSI commands, this value is ignored by the device.)

The **lun** parameter specifies the logical unit, within the device, to which the command is directed. Most often, this value is 0.

The **dirIn** parameter indicates whether data is being sent to the device (0) or being read from the device (1).

The **bank** parameter fills the Bank field of the CDB10 structure. It is usually 0.

The **lba** parameter specifies the logical block address of the device to access with this command.

The **bytes** parameter indicates the number of bytes of data being transferred.

The **data** array contains the data being sent or represents the buffer into which data will be read.

#### C# Example

```
// Create a list of devices served by the usbstor.sys driver
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC);
if (devList.Count == 0) return;

CyUSBStorDevice StorDevice = devList[0] as CyUSBStorDevice;

const byte CMD_READ    = 0x28;
byte opCode            = 0;
byte lun               = 0;
byte dirIn             = 1;
int bank               = 0;
int lba                = 0;
int xferSz             = 512;
byte [] data           = new byte[xferSz];

StorDevice.SendScsiCmd(CMD_READ, opCode, lun, dirIn, bank, lba, xferSz, data);
```

### 4.16.2 ToString()

```
public override string ToString()()  
Member of CyUSB.CyUSBStorDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

ToString returns an XML string that represents the storage device.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC);  
  
CyUSBStorDevice StorDevice = usbDevices[0] as CyUSBStorDevice;  
  
DescText.Text = StorDevice.ToString();
```

Sets DescText.Text with the following:

```
<MSC_DEVICE>  
  FriendlyName="Generic USB CF Reader USB Device"  
  Manufacturer="Compatible USB storage device"  
  Product="USB Reader"  
  SerialNumber="2004888"  
  VendorID="0x058F"  
  ProductID="0x9360"  
  Class="0x08"  
  SubClass="0x06"  
  Protocol="0x50"  
  BcdUSB="0x0100"  
</MSC_DEVICE>
```

### 4.16.3 BlockSize

```
public int BlockSize { get; }  
Member of CyUSB.CyUSBStorDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

The BlockSize property reports the size of data blocks transferred by the mass storage class device.

## 4.16.4 TimeOut

public uint **TimeOut** { set; get; }  
Member of [CyUSB.CyUSBStorDevice](#)

[Top](#) [Previous](#) [Next](#)

### Description

The TimeOut parameter maps to the TimeOutValue field of the SCSI\_PASS\_THROUGH structure that is sent to the mass storage device when [SendScsiCmd](#) is invoked.

This value is expressed in seconds and reflects how long the operating system will wait for a response from the device.

By default, this value is set to 20 in the constructor for [CyUSBStorDevice](#).

For more information on this parameter, see the the MSDN documentation for the SCSI\_PASS\_THROUGH structure.

### C# Example

```
// Create a list of devices served by the usbstor.sys driver
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_MSC);
if (devList.Count == 0) return;

CyUSBStorDevice StorDevice = devList[0] as CyUSBStorDevice;

StorDevice.TimeOut = 10;    // Set the timeout to 10 seconds
```

## 4.17 ISO\_PKT\_INFO

public struct **ISO\_PKT\_INFO**

Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

An array of ISO\_PKT\_INFO structures is passed to the [XferData](#) and [FinishDataXfer](#) methods of a [CylsocEndPoint](#) object.

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct ISO_PKT_INFO
{
    public uint Status;
    public uint Length;
}
```

## 4.18 OVERLAPPED

public struct **OVERLAPPED**

Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The OVERLAPPED structure provides a structured mapping into the operating system's event signaling structure.

The OVERLAPPED structure size is variable, depending on whether .NET is running on a 32 bit or a 64 bit CLR environment. Use [CyConst.OverlapSignalAllocSize](#) to obtain the number of bytes in this structure.

Though not passed, this structure facilitates setting-up the contents of the array, which is then passed to the [BeginDataXfer](#) and [FinishDataXfer](#) methods of the [CyUSBEndPoint](#) class.

The structure is defined in the CyUSB namespace as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct OVERLAPPED
{
    public IntPtr Internal;
    public IntPtr InternalHigh;
    public uint UnionPointerOffsetLow;
    public uint UnionPointerOffsetHigh;
    public IntPtr hEvent;
}
```

## 4.19 OverlapSignalAllocSize

```
public int OverlapSignalAllocSize { get; }
```

Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The OVERLAPPED structure size is variable, depending on whether CyUsb.NET is running in a 32-bit or 64-bit environment. Use CyConst.OverlapSignalAllocSize to obtain the number of bytes that are used internally to define the OVERLAPPED structure.

### C# Example

```
unsafe void function()
{
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice MyDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;

    byte[] overLap = new byte[CyConst.OverlapSignalAllocSize];

    fixed (byte* tmp0 = overLap)
    {
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;
        if (!MyDevice.BulkInEndPt.WaitForXfer(ovLapStatus->hEvent, 500))
        {
            MyDevice.BulkInEndPt.Abort();
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);
        }
    }
}
```

## 4.20 PInvoke

public static class **PInvoke**  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The PInvoke class is static, meaning that you need not (and cannot) create an instance of it.

PInvoke exists to expose legacy Win32 APIs that might be useful for some advanced applications.

PInvoke should only be needed when coding asynchronous data transfers using the [BeginDataXfer](#), [WaitForXfer](#) and [FinishDataXfer](#) methods of the [CyUSBEndPoint](#) class.



### 4.20.1 CreateEvent()

```
public static extern System.IntPtr CreateEvent  
( uint lpEventAttributes, uint bManualReset, uint  
bInitialState, uint lpName )  
Member of CyUSB.PInvoke
```

[Top](#) [Previous](#) [Next](#)

#### Description

CreateEvent provides the Platform Invocation for the Win32 API by the same name.

See the Microsoft Platform SDK documentation for further details about CreateEvent function.

#### C# Example

```
unsafe void function()  
{  
    byte [] overLap = new byte[CyConst.OverlapSignalAllocSize];  
  
    fixed ( byte *tmp0 = overLap )  
    {  
        OVERLAPPED *ovLapStatus = (OVERLAPPED*) tmp0;  
        ovLapStatus->hEvent = PInvoke.CreateEvent(0, 0, 0, 0);  
    }  
}
```

### 4.20.2 WaitForSingleObject( )

```
public static extern uint WaitForSingleObject ( uint  
h , uint milliseconds )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.PInvoke](#)

#### Description

WaitForSingleObject provides the Platform Invocation for the Win32 API by the same name.

See the Microsoft Platform SDK documentation for further details about WaitForSingleObject function.

#### C# Example

```
unsafe void function()  
{  
    USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
    CyUSBDevice MyDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;  
  
    byte[] overLap = new byte[CyConst.OverlapSignalAllocSize];  
  
    fixed (byte* tmp0 = overLap)  
    {  
        OVERLAPPED* ovLapStatus = (OVERLAPPED*)tmp0;  
        if (!MyDevice.BulkInEndPt.WaitForXfer(ovLapStatus->hEvent, 500))  
        {  
            MyDevice.BulkInEndPt.Abort();  
            PInvoke.WaitForSingleObject(ovLapStatus->hEvent, 500);  
        }  
    }  
}
```

## 4.21 USB\_CONFIGURATION\_DESCRIPTOR

public struct **USB\_CONFIGURATION\_DESCRIPTOR**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The USB\_CONFIGURATION\_DESCRIPTOR structure is filled-in by the [GetConfigDescriptor](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct USB_CONFIGURATION_DESCRIPTOR
{
    public byte bLength;
    public byte bDescriptorType;
    public ushort wTotalLength;
    public byte bNumInterfaces;
    public byte bConfigurationValue;
    public byte iConfiguration;
    public byte bmAttributes;
    public byte MaxPower;
}
```

## 4.22 USB\_DEVICE\_DESCRIPTOR

public struct **USB\_DEVICE\_DESCRIPTOR**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The USB\_DEVICE\_DESCRIPTOR structure is filled-in by the [GetDeviceDescriptor](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct USB_DEVICE_DESCRIPTOR
{
    public byte bLength;
    public byte bDescriptorType;
    public ushort bcdUSB;
    public byte bDeviceClass;
    public byte bDeviceSubClass;
    public byte bDeviceProtocol;
    public byte bMaxPacketSize0;
    public ushort idVendor;
    public ushort idProduct;
    public ushort bcdDevice;
    public byte iManufacturer;
    public byte iProduct;
    public byte iSerialNumber;
    public byte bNumConfigurations;
}
```

## 4.23 USB\_INTERFACE\_DESCRIPTOR

public struct **USB\_INTERFACE\_DESCRIPTOR**

Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The USB\_INTERFACE\_DESCRIPTOR structure is filled-in by the [GetInterfaceDescriptor](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct USB_INTERFACE_DESCRIPTOR
{
    public byte bLength;
    public byte bDescriptorType;
    public byte bInterfaceNumber;
    public byte bAlternateSetting;
    public byte bNumEndpoints;
    public byte bInterfaceClass;
    public byte bInterfaceSubClass;
    public byte bInterfaceProtocol;
    public byte iInterface;
}
```

## 4.24 USB\_BOS\_USB20\_DEVICE\_EXTENSION

```
public struct
USB_BOS_USB20_DEVICE_EXTENSION
Member of CyUSB
```

[Top](#) [Previous](#) [Next](#)

### Description

The USB\_BOS\_USB20\_DEVICE\_EXTENSION structure is filled-in by the [GetBosUSB20DeviceExtensionDesc](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct USB_BOS_USB20_DEVICE_EXTENSION
{
    public byte bLength;
    public byte bDescriptorType;
    public byte bDevCapabilityType;
    public uint bmAttribute;
}
```

Please refer USB3.0 specification section 9.6.2.1 for detail description of each parameter.

## 4.25 USB\_BOS\_SS\_DEVICE\_CAPABILITY

public struct **USB\_BOS\_SS\_DEVICE\_CAPABILITY**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The USB\_BOS\_SS\_DEVICE\_CAPABILITY structure is filled-in by the [GetBosSSCapabilityDescriptor](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct USB_BOS_SS_DEVICE_CAPABILITY
{
    public byte bLength;
    public byte bDescriptorType;
    public byte bDevCapabilityType;
    public byte bmAttribute;
    public ushort wSpeedsSupported;
    public byte bFunctionalitySupported;
    public byte bU1DevExitLat;
    public ushort bU2DevExitLat;
}
```

Please refer USB3.0 specification section 9.6.2.2 for detail description of each parameter.

## 4.26 USB\_BOS\_CONTAINER\_ID

public struct **USB\_BOS\_CONTAINER\_ID**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB](#)

### Description

The USB\_BOS\_CONTAINER\_ID structure is filled-in by the [GetBosContainerIDDescriptor](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct USB_BOS_CONTAINER_ID
{
    public byte bLength;
    public byte bDescriptorType;
    public byte bDevCapabilityType;
    public byte bResereved;
    unsafe public fixed byte ContainerID[CyConst.USB_BOS_CAPABILITY_TYPE_CONTAINER_ID_SIZE];
}
```

Please refer USB3.0 specification section 9.6.2.3 for detail description of each parameter.



## 4.27 USB\_BOS\_DESCRIPTOR

public struct **USB\_BOS\_DESCRIPTOR**  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The USB\_BOS\_DESCRIPTOR structure is filled-in by the [GetBosDescriptor](#) method of [CyUSBDevice](#).

The structure is defined as:

```
[StructLayout(LayoutKind.Sequential, Pack=1)]  
public struct USB_BOS_DESCRIPTOR  
{  
    public byte bLength;  
    public byte bDescriptorType;  
    public ushort wToatalLength;  
    public byte bNumDeviceCaps;  
}
```

Please refer USB3.0 specification section 9.6.2 for detail description of each parameter.

## 4.28 USBDevice

public abstract class **USBDevice** : **IDisposable**

Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The USBDevice class is abstract. That is, you cannot create an instance of this class directly. Rather, only instances of descendants of this class ( [CyUSBDevice](#), [CyUSBStorDevice](#)) can be instantiated.

However, the fact that the class is abstract allows grouping of different descendant objects in a single data structure. For instance, the [USBDeviceList](#) class maintains a list of USBDevice objects. Each object in that list is actually an instance of either [CyUSBDevice](#) or [CyUSBStorDevice](#).

This abstract, parent class contains several data members which are common to all its descendants. These can be accessed from a general USBDevice object that has been assigned to a true, instantiated object of one of the descendant classes.

### C# Example

```
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
// Create a list of devices served by the cyusb3.sys driver  
  
// Get the FriendlyName of the first object, regardless of its class  
USBDevice device = devList[0];  
string fName = device.FriendlyName;
```

### 4.28.1 Dispose()

public void **Dispose()**  
Member of [CyUSB.USBDevice](#)

[Top](#) [Previous](#) [Next](#)

#### Description

In order to support the IDisposable interface, USBDevice implements the Dispose method.

You should never invoke the Dispose method of a USBDevice directly. Rather, the appropriate technique is to call the Dispose method of the [USBDeviceList](#) object that contains the USBDevice objects.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList usbDevices;
```

```
public Form1()
{
    InitializeComponent();

    App_PnP_Callback evHandler = new App_PnP_Callback(PnP_Event_Handler);

    usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB | CyConst.DEVICES_HID | CyConst.DEVICES_MSC,
    evHandler);
}
```

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (usbDevices != null) usbDevices.Dispose();
}
```

### 4.28.2 Equals()

```
public override bool Equals(object right)  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

The Equals method allows the comparison of two USBDevice objects (or their descendants) to determine if they represent the same physical USB device.

If the [Path](#) string for two devices are identical, Equals returns true. Otherwise, it returns false.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
// Uses the Equals method to determine if dev is already in the list  
public byte DeviceIndex(USBDevice dev)  
{  
    byte x = 0; // Index of tmp  
  
    foreach (USBDevice tmp in Items)  
    {  
        if (dev.Equals(tmp))  
            return x;  
  
        x++;  
    }  
  
    return 0xFF; // Device wasn't found  
}
```

### 4.28.3 BcdUSB

```
public ushort BcdUSB { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the value of the **bcdUSB** field of the device's USB descriptor.

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the BcdUSB of the first object  
USBDevice device = devList[0];  
UInt16 bcd = device.BcdUSB;
```

## 4.28.4 DevClass

```
public ushort DevClass { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

This property reports the value of the **bDeviceClass** field from the device's Device Descriptor.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the DevClass of the first object  
USBDevice device = devList[0];  
UInt16 dClass = device.DevClass;
```

## 4.28.5 DevProtocol

```
public byte DevProtocol { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

This property reports the value of the device descriptor's **bDeviceProtocol** field.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the DevProtocol of the first object  
USBDevice device = devList[0];  
byte protocol = device.DevProtocol;
```

### 4.28.6 DevSubClass

```
public byte DevSubClass { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property reports the value of the device descriptor's **bDeviceSubClass** field.

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the DevSubClass of the first object  
USBDevice device = devList[0];  
byte subclass = device.DevSubClass;
```



### 4.28.7 DriverName

```
public string DriverName { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

DriverName returns an upper-case string that represents the USB device driver serving the USBDevice. This value will be one of the following:

cyusb3.sys

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the DriverName of the first object  
USBDevice device = devList[0];  
string sDriver = device.DriverName;
```

### 4.28.8 FriendlyName

```
public string FriendlyName { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

FriendlyName returns the device description string supplied by the driver's .inf file.

To locate a device having a particular FriendlyName, see the [USBDeviceList indexer](#) methods.

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the FriendlyName of the first object  
USBDevice device = devList[0];  
string fName = device.FriendlyName;
```

## 4.28.9 Manufacturer

```
public string Manufacturer { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

Manufacturer returns the string indicated by the device descriptor's **iManufacturer** field.

To locate a device from a particular Manufacturer, see the [USBDeviceList indexer](#) methods.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the Manufacturer of the first object  
USBDevice device = devList[0];  
string mfg = device.Manufacturer;
```

## 4.28.10 Name

```
public string Name { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

Name returns the product string from the device descriptor's **iProduct** field.

The [Product](#) and Name members of USBDevice should always be identical.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the Name of the first object  
USBDevice device = devList[0];  
string DeviceName = device.Name;
```

## 4.28.11 Path

```
public string Path { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

Path returns the Windows system string used to obtain a Windows handle to the device.

In typical use of the library, this value should never be needed. It is exposed as a "just in case" hook for debugging purposes or advanced techniques that would circumvent the CyUSB API.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the Path of the first object  
USBDevice device = devList[0];  
string DevicePath = device.Path;
```

## 4.28.12 Product

```
public string Product { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

Product returns the string indicated by the device descriptor's **iProduct** field.

The Product and [Name](#) members of USBDevice should always be identical.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the Product of the first object  
USBDevice device = devList[0];  
string ProductName = device.Product;
```

### 4.28.13 ProductID

```
public ushort ProductID { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

#### Description

This property returns the value of the device descriptor's **idProduct** field.

To locate a device having a particular ProductID, see the [USBDeviceList indexer](#) methods.

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the ProductID of the first object  
USBDevice device = devList[0];  
UInt16 PID = device.ProductID;
```

## 4.28.14 SerialNumber

```
public string SerialNumber { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

SerialNumber returns the string indicated by the device descriptor's **iSerialNumber** field.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the SerialNumber of the first object  
USBDevice device = devList[0];  
string SerNum = device.SerialNumber;
```



## 4.28.15 Tree

public virtual System.Windows.Forms.TreeNode **Tree**

{ get; }

Member of [CyUSB.USBDevice](#)

[Top](#) [Previous](#) [Next](#)

### Description

The Tree property returns a Windows.Forms.TreeNode.

The Text property of the TreeNode is the string returned by the [FriendlyName](#) property.

The TreeNode of this base class implementation has no child nodes. However, the TreeNodes returned by descendants of USBDevice usually do have child nodes.

The Tag property of the returned TreeNode contains a reference to the USBDevice object (*this*).

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void RefreshDeviceTree()
{
    DeviceTreeView.Nodes.Clear();
    DescText.Text = "";

    foreach (USBDevice dev in usbDevices)
        DeviceTreeView.Nodes.Add(dev.Tree);
}

private void DeviceTreeView_AfterSelect(object sender, TreeViewEventArgs e)
{
    TreeNode selNode = DeviceTreeView.SelectedNode;
    DescText.Text = selNode.Tag.ToString();
}
```

## 4.28.16 USBAddress

```
public byte USBAddress { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

USBAddress returns the bus address of the device.

This is the address value used by the Windows USBDI stack. It is not particularly useful at the application level.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the USBAddress of the first object  
USBDevice device = devList[0];  
byte addrUSB = device.USBAddress;
```

## 4.28.17 VendorID

```
public ushort VendorID { get; }  
Member of CyUSB.USBDevice
```

[Top](#) [Previous](#) [Next](#)

### Description

This property returns the value of the device descriptor's **idVendor** field.

To locate a device having a particular VendorID, see the [USBDeviceList indexer](#) methods.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList devList = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (devList.Count == 0) return;  
  
// Get the ProductID of the first object  
USBDevice device = devList[0];  
UInt16 VID = device.VendorID;
```

## 4.29 USBDeviceList

public class **USBDeviceList** : IDisposable,  
IEnumerable  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The USBDeviceList class is at the heart of the CyUSB class library. In order to successfully utilize the library, a good working knowledge of the USBDeviceList class is essential.

USBDeviceList represents a dynamic list of USB devices that are accessible via the class library. When an instance of USBDeviceList is created, it populates itself with [USBDevice](#) objects representing all the USB devices served by the indicated device selector mask. These USBDevice objects have all been properly initialized and are ready for use.

Once an instance of the USBDeviceList class has been constructed, the USBDeviceList [index operators](#) make it easy to locate a particular device and begin using it.

Because USBDeviceList implements the IDisposable interface, you should call its [Dispose](#) method when you finish using a USBDeviceList object.

Because USBDeviceList implements the IEnumerable interface, you iterate through a USBDeviceList object's items using the *foreach* keyword.

### C# Example 1

```
// Create a list of devices served by the cyusb3.sys driver
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
if (usbDevices.Count == 0) return;

// Get the first device in the list
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;

// Get the first device having FriendlyName == "My USB Device"
myDev = usbDevices["My USB Device"] as CyUSBDevice;

// Get the first device having VendorID == 0x04B4 and ProductID == 0x8613
myDev = usbDevices[0x04B4, 0x8613] as CyUSBDevice;

if (myDev != null)
{
    byte altSetting = myDev.AltIntfc;
}
```

### C# Example 2

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList usbDevices;
```

```
public Form1()
{
    InitializeComponent();
}
```

```
usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB | CyConst.DEVICES_HID | CyConst.DEVICES_MSC);
usbDevices.DeviceAttached += new EventHandler(usbDevices_DeviceAttached);
usbDevices.DeviceRemoved += new EventHandler(usbDevices_DeviceRemoved);
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (usbDevices != null) usbDevices.Dispose();
}

void usbDevices_DeviceRemoved(object sender, EventArgs e)
{
    RefreshDeviceTree();
}

void usbDevices_DeviceAttached(object sender, EventArgs e)
{
    RefreshDeviceTree();
}

private void RefreshDeviceTree()
{
    DeviceTreeView.Nodes.Clear();

    foreach (USBDevice dev in usbDevices)
        DeviceTreeView.Nodes.Add(dev.Tree);
}
```

### 4.29.1 DeviceAttached( )

public event System.EventHandler **DeviceAttached**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.USBDeviceList](#)

#### Description

When a new USB device is plugged-in to the bus, the connection event can be detected and some action can be taken.

Detection of the event is automatically set-up by the USBDeviceList object.

Handling of the event requires that an EventHandler object be assigned to the DeviceAttached event handler.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList usbDevices;

public Form1()
{
    InitializeComponent();

    usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB | CyConst.DEVICES_MSC);
    usbDevices.DeviceAttached += new EventHandler(usbDevices_DeviceAttached);
}

void usbDevices_DeviceAttached(object sender, EventArgs e)
{
    USBEventArgs usbEvent = e as USBEventArgs;
    // Take some action
}
```

## 4.29.2 DeviceRemoved()

public event System.EventHandler **DeviceRemoved**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.USBDeviceList](#)

### Description

When a USB device is disconnected from the bus, the removal event can be detected and some action can be taken.

Detection of the event is automatically set-up by the USBDeviceList object.

Handling of the event requires that an EventHandler object be assigned to the DeviceRemoved event handler.

### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList usbDevices;
```

```
public Form1()
{
    InitializeComponent();

    usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB | CyConst.DEVICES_MSC);
    usbDevices.DeviceRemoved += new EventHandler(usbDevices_DeviceRemoved);
}
```

```
void usbDevices_DeviceRemoved(object sender, EventArgs e)
{
    USBEventArgs usbEvent = e as USBEventArgs;
    // Take some action
}
```

### 4.29.3 Dispose()

public void **Dispose()**  
Member of [CyUSB.USBDeviceList](#)

[Top](#) [Previous](#) [Next](#)

#### Description

In order to support the IDisposable interface, USBDeviceList implements the Dispose method.

You should invoke Dispose when you finish using a USBDeviceList object.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
USBDeviceList usbDevices;
```

```
public Form1()  
{  
    InitializeComponent();  
  
    usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB | CyConst.DEVICES_HID | CyConst.DEVICES_MSC);  
}
```

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)  
{  
    if (usbDevices != null) usbDevices.Dispose();  
}
```



#### 4.29.4 USBDeviceList()

public **USBDeviceList** ( byte *DeviceMask*)  
Member of [CyUSB.USBDeviceList](#)

[Top](#) [Previous](#) [Next](#)

##### Description

This constructor creates a USBDeviceList object and populates it with USBDevice objects. The USBDevice objects in the list are those indicated by the DeviceMask parameter.

##### Parameters

System.Byte *DeviceMask*

This parameter specifies the subset of USB devices that will be represented in the DeviceList. The subset is defined by performing a bitwise OR of the following device constants:

CyConst.[DEVICES\\_CYUSB](#)

CyConst.[DEVICES\\_MSC](#)

CyConst.[DEVICES\\_HID](#)

##### Return Value

Returns a USBDeviceList object that has been populated with USBDevice objects.

##### C# Example

```
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
```

## 4.29.5 Count

public int **Count** { get; }  
Member of [CyUSB.USBDeviceList](#)

[Top](#) [Previous](#) [Next](#)

### Description

The Count property reflects the number of USBDevice objects in the USBDeviceList.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;
```

## 4.29.6 USBDeviceList [int index]

public const CyUSB.USBDevice **this** [ int *index* ]  
Member of [CyUSB.USBDeviceList](#)

[Top](#) [Previous](#) [Next](#)

### Description

This index operator provides access to elements of the USBDeviceList using standard array integer indexing.

### Parameters

int *index*

*index* refers to the numerical order of the item in the USBDeviceList.

### Return Value

Returns a [USBDevice](#) object. Because USBDevice is an abstract class, the object returned will need to be casted into a [CyUSBDevice](#) or a [CyUSBStorDevice](#) to be of much use.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
if (usbDevices.Count == 0) return;

// Get the first device in the list
CyUSBDevice myDev = usbDevices[0] as CyUSBDevice;

if (myDev != null)
{
    byte altSetting = myDev.AltIntfc;
}
```

### 4.29.7 USBDeviceList [string fName]

```
public const CyUSB.USBDevice this [ string  
    FriendlyName ]  
Member of CyUSB.USBDeviceList
```

[Top](#) [Previous](#) [Next](#)

#### Description

This index operator provides access to elements of the USBDeviceList based on the [FriendlyName](#) property of the [USBDevice](#) objects in the list.

#### Parameters

string *FriendlyName*

*FriendlyName* is a string that will be compared to the [FriendlyName](#) property of the devices in the list in order to locate a particular device.

#### Return Value

Returns the first [USBDevice](#) object that matches the *FriendlyName* . Because USBDevice is an abstract class, the object returned will need to be casted into a [CyUSBDevice](#) or a [CyUSBStorDevice](#) to be of much use.

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Get the first device having FriendlyName == "My USB Device"  
CyUSBDevice myDev = usbDevices["My USB Device"] as CyUSBDevice;  
  
if (myDev != null)  
{  
    byte altSetting = myDev.AltIntfc;  
}
```

## 4.29.8 USBDeviceList [int VID, int PID]

```
public const CyUSB.USBDevice this [ int VendorID,  
int ProductID ]  
Member of CyUSB.USBDeviceList
```

[Top](#) [Previous](#) [Next](#)

### Description

This index operator provides access to elements of the USBDeviceList based on the [VendorID](#) and [ProductID](#) properties of the [USBDevice](#) objects in the list.

### Parameters

int *VendorID*

*VendorID* will be compared to the [VendorID](#) property of the devices in the list in order to locate a particular device.

int *ProductID*

*ProductID* will be compared to the [ProductID](#) property of the devices in the list in order to locate a particular device.

### Return Value

Returns the first [USBDevice](#) object that matches both the *VendorID* and *ProductID*. Because USBDevice is an abstract class, the object returned will need to be casted into a [CyUSBDevice](#) or a [CyUSBStorDevice](#) to be of much use.

### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Get the first device having VendorID == 0x04B4 and ProductID == 0x8613  
CyUSBDevice myDev = usbDevices[0x04B4, 0x8613] as CyUSBDevice;  
  
if (myDev != null)  
{  
    byte altSetting = myDev.AltIntfc;  
}
```

### 4.29.9 USBDeviceList [string sMfg, string sProd]

```
public const CyUSB.CyHidDevice this [ int VID, int  
PID ]
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.USBDeviceList](#)

#### Description

This index operator provides access to elements of the USBDeviceList based on the [Manufacturer](#) and [Product](#).

#### Parameters

string *Manufacturer*

*Manufacturer* will be compared to the [Manufacturer](#) property of the devices in the list in order to locate a particular device.

string *Product*

*Product* will be compared to the [Product](#) property of the devices in the list in order to locate a particular device.

#### Return Value

Returns the first [USBDevice](#) object that matches both the *Manufacturer* and *Product* properties. Because USBDevice is an abstract class, the object returned will need to be casted into a [CyUSBDevice](#) or a [CyUSBStorDevice](#) to be of much use.

#### C# Example

```
// Create a list of devices served by the cyusb3.sys driver  
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);  
if (usbDevices.Count == 0) return;  
  
// Get the first device having Manufacturer == "Cypress" and Product == "NX2LP"  
CyHidDevice myDev = usbDevices["Cypress","Fx2LP"] as CyHidDevice;
```

## 4.30 Util

public static class **Util** : System.Object  
Member of [CyUSB](#)

[Top](#) [Previous](#) [Next](#)

### Description

The Util class encapsulates a group of static methods that provide various useful functions.

Because the methods are declared [static](#), no Util object is needed to invoke the methods.

### 4.30.1 ParseHexData()

```
public static bool ParseHexData ( System.
Collections.ArrayList rawList , byte[] FwBuf , ref
ushort FwLen , ref ushort FwOff )
Member of CyUSB.Util
```

[Top](#) [Previous](#) [Next](#)

#### Description

ParseHexData consumes an ArrayList of strings representing the lines of text from a .hex file. It creates a byte array image of the firmware specified by the list, with all code bytes at the specified locations in the array. ParseHexData is called by [ParseHexFile](#).

#### Parameters

System.Collections.ArrayList *rawList*

An array of strings representing the lines of text from an Intel .hex file.

byte[] *FwBuf*

An array of bytes that will hold the parsed firmware code bytes from the *rawList*. When ParseHexData finishes, this array contains all the code bytes placed in proper sequence within the array.

Before filling the array with code bytes from the *rawList*, each byte of *FwBuf* is initialized to 0xFF.

*FwBuf* should be [MAX\\_FW\\_SIZE](#) in length.

System.UInt16 *FwLen*

When ParseHexData finishes, *FwLen* contains the offset (i.e address) of the last valid data byte in the image.

System.UInt16 *FwOff*

When ParseHexData finishes, *FwOff* contains the offset (i.e address) of the first valid data byte in the image.

#### Return Value

Returns [false](#) if the *rawList* defines any bytes to be placed at an offset greater than [MAX\\_FW\\_SIZE](#). Otherwise, returns [true](#).

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void GetFwImage()
{
    String FwFile = "myfirmware.hex";
    byte[] FwImage = new byte[Util.MaxFwSize];
```



```
    ushort ImageLen    = 0;
    ushort Fw Offset    = 0;

    ArrayList codeLines = new ArrayList();

    TgtDevice.FillCodeList(Fw File,codeLines);

    bool bParsed = Util.ParseHexData(codeLines,Fw Image, ref ImageLen, ref Fw Offset);
}
```

### 4.30.2 ParseHexFile( )

```
public static bool ParseHexFile ( string fName , byte
[] FwBuf , ref ushort FwLen , ref ushort FwOff )
Member of CyUSB.Util
```

[Top](#) [Previous](#) [Next](#)

#### Description

ParseHexFile consumes an Intel .hex formatted ASCII file and creates a byte array image of the firmware code specified by the file, with all code bytes at the specified locations in the array.

#### Parameters

System.String *fName*

The name of the Intel .hex formatted ASCII file to be parsed. The filename should include the relative or full directory path for the file.

byte[] *FwBuf*

An array of bytes that will hold the parsed code bytes from the .hex file. When ParseHexFile finishes, this array contains all the code bytes placed in proper sequence within the array.

Before filling the array with code bytes from the .hex file, each byte of *FwBuf* is initialized to 0xFF.

*FwBuf* should be [MAX\\_FW\\_SIZE](#) in length.

System.UInt16 *FwLen*

When ParseHexFile finishes, *FwLen* contains the offset (i.e address) of the last valid code byte in the image.

System.UInt16 *FwOff*

When ParseHexFile finishes, *FwOff* contains the offset (i.e address) of the first valid code byte in the image.

#### Return Value

Returns [false](#) if the .hex file defines any bytes to be placed at an offset greater than [MAX\\_FW\\_SIZE](#). Otherwise, returns [true](#).

#### C# Example

```
private void GetFwImage()
{
    String FwFile      = "myfirmware.hex";
    byte [] FwImage    = new byte[Util.MaxFwSize];

    ushort ImageLen    = 0;
    ushort FwOffset    = 0;
```

---

```
    bool bParsed = Util.ParseHexFile(Fw File, Fw Image, ref ImageLen, ref Fw Offset);  
}
```

### 4.30.3 ParseIICData( )

```
public static bool ParseIICData( ) ( byte[] fData,  
byte[] FwBuf, ref ushort FwLen, ref ushort FwOff )  
Member of CyUSB.Util
```

[Top](#) [Previous](#) [Next](#)

#### Description

ParseIICData consumes an array of bytes containing the data from an .iic file. It creates a byte array image of the data specified by the file, with all firmware code bytes at the specified locations in the array.

ParseIICData is called by [ParseIICFile](#).

#### Parameters

byte[] *fData*

An array containing the contents of an .iic file.

byte[] *FwBuf*

An array of bytes that will hold the parsed data from the *fData*. When ParseIICData finishes, this array contains all the firmware code bytes placed in proper sequence.

Before filling the array with code bytes from the *fData*, each byte of *FwBuf* is initialized to 0xFF.

*FwBuf* should be [MAX\\_FW\\_SIZE](#) in length.

System.UInt16 *FwLen*

When ParseIICData finishes, *FwLen* contains the offset (i.e address) of the last valid data byte in the image.

System.UInt16 *FwOff*

When ParseIICData finishes, *FwOff* contains the offset (i.e address) of the first valid data byte in the image.

#### Return Value

Returns **false** if the *fData* defines any bytes to be placed at an offset greater than [MAX\\_FW\\_SIZE](#). Otherwise, returns **true**.

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private bool LoadFX2FWToRAM()  
{  
    USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
    CyUSBDevice       CyDevice     = usbDevices[0] as CyUSBDevice;
```

```

    ushort ImageLen    = 0;
    ushort Fw Offset   = 0;

    // FwBuf holds the file contents, suitable for the EEPROM
    // Now, parse it into FwImage, putting each record at the right offset,
    // suitable for the FX2 RAM
    byte[] Fw Image= new byte[Util.MaxFw Size];
    Util.ParseIICData(Fw Buf, Fw Image, ref ImageLen, ref Fw Offset);

    ResetFX2(1); // Halt

    ushort chunk = 2048;
    byte [] buffer = new byte[chunk];

    CyControlEndPoint ep0 = CyDevice.ControlEndPt;

    for (ushort i=Fw Offset; i<ImageLen; i+=chunk)
    {
        ep0.Value = i;
        int len = ((i + chunk)<ImageLen) ? chunk : ImageLen - i;
        Array.Copy(Fw Image,i,buffer,0,len);

        ep0.Write(ref buffer, ref len);
    }

    ResetFX2(0); // Run

    return true;
}

private void ResetFX2(byte hold)
{
    USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);
    CyUSBDevice      CyDevice      = usbDevices[0] as CyUSBDevice;

    if (CyDevice == null) return;

    byte[] dta = new byte[8];

    CyControlEndPoint ep0 = CyDevice.ControlEndPt;

    ep0.Target        = CyConst.TGT_DEVICE;
    ep0.RegType        = CyConst.REQ_VENDOR;
    ep0.Value          = 0xE600;
    ep0.Index          = 0x0000;

    ep0.RegCode        = 0xA0;
    dta[0]              = hold;
    int len            = 1;

    ep0.Write(ref dta, ref len);

    //Thread.Sleep(500); //Wait for some time
}

```

#### 4.30.4 ParseIICFile( )

```
public static bool ParseIICFile ( System.String
    fName , byte[] FwBuf , ref ushort FwLen , ref
    ushort FwOff )
Member of CyUSB.Util
```

[Top](#) [Previous](#) [Next](#)

##### Description

ParseIICFile consumes an .iic firmware file and creates a byte array image of the firmware code specified by the file, with all code bytes at the specified locations in the array.

##### Parameters

System.String *fName*

The name of the .iic file to be parsed. The filename should include the relative or full directory path for the file.

byte[] *FwBuf*

An array of bytes that will hold the parsed code bytes from the .iic file. When ParseIICFile finishes, this array contains all the code bytes placed in proper sequence within the array.

Before filling the array with code bytes from the .iic file, each byte of *FwBuf* is initialized to 0xFF.

*FwBuf* should be [MAX\\_FW\\_SIZE](#) in length.

System.UInt16 *FwLen*

When ParseIICFile finishes, *FwLen* contains the offset (i.e address) of the last valid code byte in the image.

System.UInt16 *FwOff*

When ParseIICFile finishes, *FwOff* contains the offset (i.e address) of the first valid code byte in the image.

##### Return Value

Returns [false](#) if the .iic file defines any bytes to be placed at an offset greater than [MAX\\_FW\\_SIZE](#). Otherwise, returns [true](#).

##### C# Example

```
private void GetFwImage()
{
    String FwFile = "myfirmware.iic";
    byte[] FwImage = new byte[Util.MaxFwSize];

    ushort ImageLen = 0;
    ushort FwOffset = 0;
```

```
bool bParsed = Util.ParseIICFile(Fw File, Fw Image, ref ImageLen, ref Fw Offset);  
}
```

### 4.30.5 ReverseBytes( )

```
public static int ReverseBytes ( byte* dta, int  
bytes )
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.Util](#)

#### Description

This method is used to reverse the byte-order of a 2-byte or 4-byte integer.

#### Parameters

byte \**dta*

A pointer to the first byte of the value to be reversed.

int *bytes*

The number of bytes comprising the value to be reversed. Acceptable values for this parameter are 2 and 4.

#### Return Value

Returns a 4-byte signed integer (int) value represented by the reversed bytes.



### 4.30.6 ReverseBytes( )

```
public static int ReverseBytes(byte[] dta, int xStart  
, int bytes)  
Member of CyUSB.Util
```

[Top](#) [Previous](#) [Next](#)

#### Description

This method is used to reverse the the order of a sequence of bytes contained in an array of bytes.

#### Parameters

byte[] *dta*

An array of bytes to be reversed.

int *xStart*

The index in the *dta* array of the first byte in the sequence to be reversed.

int *bytes*

The number of bytes comprising the value to be reversed. Any number of bytes within the dimensions of the *dta* array can be reversed.

#### Return Value

Returns a 4-byte signed integer (System.Int32) value represented by the reversed bytes.

### 4.30.7 Assemblies

```
public static string Assemblies { get; }
```

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.Util](#)

#### Description

The Assemblies property returns a formatted list of an application's assemblies and the version numbers for the assemblies.

System and mscorlib assemblies are not reported.

#### Return Value

The returned string contains a header and a \r\n delimited list of assemblies. Each assembly name is followed by one or more tab characters and the assembly's version number, as shown here.

```
ASSEMBLY\t\t\tVERSION\r\n\r\nAssembly1\t\t\tAssembly1Version\r\nAssembly2\t\t\tAssembly2Version\r\nAssemblyN\t\t\tAssemblyNVersion\r\n\r\n
```

#### C# Example

NOTE : This is not a ready to compile code, you can use this sample code as a guideline.

```
private void AboutMenuItem_Click(object sender, System.EventArgs e)
{
    string assemblyList = Util.Assemblies();
    MessageBox.Show (assemblyList,Text);
}
```

### 4.30.8 MaxFwSize

```
public static ushort MaxFwSize { set; get; }
```

[Top](#) [Previous](#) [Next](#)

#### Description

MaxFwSize represents the maximum address space of a memory device and is used by the [ParseHexData](#), [ParseHexFile](#) and [ParseIICFile](#) methods.

It's default value is 0x4000 (or 16,384).

## 4.31 XMODE

public enum **XMODE**

[Top](#) [Previous](#) [Next](#)

Member of [CyUSB.CyConst](#)

### Description

XMODE is an enumeration containing the values BUFFERED and DIRECT. These can be used to set the [XferMode](#) property of a [CyUSBEndPoint](#) object.

The API would create a temporary buffer to pass to the driver, then copy the user's data to/from that buffer. This double buffering scheme incurred a performance penalty and was replaced by the more efficient direct transfer mode.

In direct transfer mode, the API passes the user's buffer to the driver and the driver accesses that buffer directly.

Normally you will want to use XMODE.DIRECT, rather than XMODE.BUFFERED, as the direct transfer method is faster. XMODE.BUFFERED exists, primarily, for internal compatibility testing and debugging purposes.

The value of XMODE.BUFFERED is 1.

[CyUSBEndPoint](#) objects have their [XferMode](#) property set to XMODE.DIRECT by default.

### C# Example

```
USBDeviceList    usbDevices    = new USBDeviceList(CyConst.DEVICES_CYUSB);  
CyUSBDevice      StreamDevice  = usbDevices["Cy Stream Device"] as CyUSBDevice;  
  
if (StreamDevice != null)  
    StreamDevice.BulkInEndPt.XferMode = XMODE.BUFFERED;
```

## 5 Features Not Supported

The Following features are not supported by the C# library, CyUSB.dll.

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. Set/Get Transfer size

The [XferSize](#) member variable of the [CyUSBEndpoint](#) to set/get the transfer size of endpoint is no longer supported. Please refer the [XferSize](#) for more information on it.

5. CyUSB DLL does not support HID API.

# Index

## - D -

### Data Transfers

Synchronous Transfers 188

### Descriptors

Configuration 123

Device 124

Endpoint 173

Interface 209

Listing Descriptor Contents 99

### Devices

Finding USB Devices 121, 268, 275, 276, 277, 278

Manufacturer 259

Product 262

Serial Number 264

## - E -

### Endpoints 165

Bulk Endpoints 163

Control Endpoints 164

Interrupt Endpoints 169

Isochronous Endpoints 171

## - U -

uint 246

USBDeviceList 268