Cypress CyUSB .NET DLL Programmer's Reference

Table of Contents

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 Stuido.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 *CyUSB.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 HID or Mass Storage 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 below example C# code 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 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 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:

- 64-Bit Platform Support
- API Change Notice
- New API

3.1 64-Bit Platform Support

64-Bit Platform Support

Top Previous Next

Description

This release of *CyUSB.dll* is compatible with .Net 2.0 Projects that are built for the following platform targets:

- "Any CPU"
- "x86"
- "x64"

CyUSB.dll is now capable of running on both 32-bit and 64-bit platforms, and will interface with 32-bit and 64-bit versions of Cypress Semiconductor's "CYUSB.sys" Windows(tm) USB device driver, Microsoft's USB HID device driver and Microsoft's USB mass storage driver(usbstor.sys).

Some aspects of the API were changed in order to add 64-bit support. Please see API Change Notice for details.

3.2 API Change Notice

API Change Notice

Top Previous Next

Description

The OVERLAPPED structure provides a structured mapping into the operating system's event signaling structure. Through the release of CyUsb version 2.2007.46.4, memory could be allocated for the overlapped buffer by hard-coding as follows.

```
byte [] overLap       = new  byte[20];
```

See the revised help for OVERLAPPED, as well as the new property variable, OverlapSignalAllocSize for details.

3.3 New API

1. CyConst.SetClassGuid()

The SetClassGuid() method is a member of CyConst class.

This method should be called to set the new class GUID,if one uses vendor specific device class GUID.

The input parameter to this API is a string, which contain the new class GUID.

C# Example:

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

```
public Form1() //application starts here.
 {
    Initialize();

//Initializes form resources
    InitializeComponent();

//Set the customer class GUID(vendor specific)
CyConst.SetClassGuid("{CDBF8987-75F1-468e-8217-97197F88F773}");

    ..............
       ..............
       ......

    Form1_Resize(this, null);


 }
```

2. `public unsafe bool XferData(ref byte[] buf, ref int len, bool PacketMode)`

XferData() method of CyUSBEndPoint is overloaded such that it can be used for partial IN transfer on Bulk/Interrupt endpoints.

if PacketMode = true, partial data transfer is enabled or else calls the `bool XferData(ref byte[] buf, ref int len)` where partial data is discarded.

This parameter has no effect on Out endpoint.

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, false);
        }
        else if (MyDevice.BulkInEndPt != null)
         {
            int len = 512;
                byte [] buf = new byte[len];
                   MyDevice.BulkInEndPt.XferData(ref buf, ref len, true);
         }
}
```

3. public bool LoadExternalRam(string fwFile)

Used to load the firmware which contains external RAM address into the external RAM.

C# Example

```
USBDeviceList       usbDevices       = new USBDeviceList(CyConst.DEVICES_CYUSB);
CyFX2Device MyDevice = usbDevices[0x04B4,0x1003] as CyFX2Device;

if (MyDevice != null)
 {
       MyDevice.LoadExternalRam("BulkLoop.hex");
 }
```

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 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 BulkInEndPt and BulkOutEndPt.

C# Example

```
// Find a bulk IN endpoint in the EndPoints[] array
CyBulkEndPoint BulkIn = null;

// Create a list of devices served by CyUSB.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;
```

4.2 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.

4.2.1 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 CyUSB.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 CyUSB.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);

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

C# Example 2

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

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

4.2.2 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 CyUSB.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_HID | CyConst.DEVICES_CYUSB);

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

4.2.3 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 CyUSB.sys
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_MSC | CyConst.DEVICES_CYUSB);

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

4.2.4 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.ReqType     = CyConst.REQ_VENDOR;
    CtrlEndPt.Direction = CyConst.DIR_FROM_DEVICE;
    CtrlEndPt.ReqCode     = 0xB0;
    CtrlEndPt.Value       = 0;
    CtrlEndPt.Index       = 0;

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

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

```
}
```

4.2.5 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.ReqType     = CyConst.REQ_VENDOR;
      CtrlEndPt.Direction = CyConst.DIR_TO_DEVICE;
      CtrlEndPt.ReqCode     = 0xC0;
      CtrlEndPt.Value       = 2;
      CtrlEndPt.Index       = 0;

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

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

4.2.6 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);
}
```

4.2.7 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);
    }
```

4.2.8 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);
    }
```

4.2.9 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 lnEndpt;


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


    if (lnEndpt != 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);
         }

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

   }
```

4.2.10 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.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);
}
```

4.2.11 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);
    }
```

4.2.12 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);
    }
```

4.2.13 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.ReqType     = CyConst.REQ_VENDOR;
      CtrlEndPt.Direction = CyConst.DIR_TO_DEVICE;
      CtrlEndPt.ReqCode     = 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);
}
```

4.2.14 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);
      }
   }
}
```

4.3 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);
}
```

4.3.1 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.ControlEndPt;

if (CtrlEndPt != null)
{
        CtrlEndPt.Target      = CyConst.TGT_DEVICE;
        CtrlEndPt.ReqType     = CyConst.REQ_VENDOR;
        CtrlEndPt.ReqCode     = 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);
    }
```

4.3.2 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.ReqType     = CyConst.REQ_VENDOR;
    CtrlEndPt.ReqCode     = 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);
}
```

4.3.3 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.ReqType      = CyConst.REQ_VENDOR;
      CtrlEndPt.Direction = CyConst.DIR_TO_DEVICE;
      CtrlEndPt.ReqCode      = 0xC0;
      CtrlEndPt.Value        = 2;
      CtrlEndPt.Index        = 0;

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

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

4.3.4 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       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      = 0xC0;
      CtrlEndPt.Value        = 2;
      CtrlEndPt.Index        = 0;

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

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

4.3.5 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      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);
}
```

4.3.6 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      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);
}
```

4.3.7 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.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);
    }
```

4.3.8 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      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);
}
```

4.3.9 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.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;            // Specifies the Configuration Descriptor
      CtrlEndPt.Index      = 0;

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

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

4.4 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");
```

4.4.1 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();
            }
        }
```

4.4.2 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();
        }
    }
```

4.4.3 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
    w hile (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);

}
```

4.5 CyHidButton

public class **CyHidButton** Top Previous Next

Member of CyUSB

Description

CyHidButton represents USB Human Interface Devices Button data item. It contains the following read-only properties which reflect the HID button descriptor values:

```
public ushort BitField { get; }
public ushort DataIndex { get; }
public ushort DataIndexMax { get; }
public ushort DesignatorIndex { get; }
public ushort DesignatorIndexMax { get; }
public bool IsAbsolute { get; }
public bool IsAlias { get; }
public bool IsDesignatorRange { get; }
public bool IsRange { get; }
public bool IsStringRange { get; }
public ushort LinkCollection { get; }
public ushort LinkUsage { get; }
public ushort LinkUsagePage { get; }
public ushort ReportID { get; }
public ushort StringIndex { get; }
public ushort StringMax { get; }
public ushort Usage { get; }
public ushort UsageMax { get; }
public ushort UsagePage { get; }
```

4.6 CyHidDevice

public class **CyHidDevice** : CyUSB.USBDevice Top Previous Next

Member of CyUSB

Description

CyHidDevice represents USB Human Interface Devices (HID) such as mice and keyboards.

Because CyHidDevice is a descendant of USBDevice, it inherits all the members of USBDevice.

NOTE: USB HID descriptors and organization are complex. The CyHidDevice abstraction does not present a complete or exhaustive model for HID devices. It exists to provide basic support for communicating with and identifying HID devices.

4.6.1 GetFeature( )

public bool **GetFeature** (int *rptID*) Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Features represent configuration data for a HID device.

GetFeature reads RptByteLen bytes, corresponding to the *rptID* report, for a HID Feature. The bytes transferred from the device are found in the Features.DataBuf.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
  CyHidDevice hidAnalyzer = HidDevices[0] as CyHidDevice;
byte[] Raw Data = new byte[hidAnalyzer.Features.RptByteLen];

if ((hidAnalyzer != null) && hidAnalyzer.GetFeature(0))
   for (int x = 1; x < hidAnalyzer.Features.RptByteLen; x++)
        Raw Data[x++] = hidAnalyzer.Features.DataBuf[x];
```

4.6.2 GetInput( )

public bool **GetInput** (int *rptID*) Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Inputs provide access to read-only HID controls (buttons and values).

GetInput uses the Win32 HidD_GetInputReport( ) function to read RptByteLen bytes from the device using the Control endpoint.

rptID specifies the ReportID to read. The bytes transferred from the device are found in the Inputs.DataBuf.

NOTE: GetInput is only supported under WindowsXP and newer.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

string s = "";

if (hidDev.GetInput(hidDev.Inputs.ID))
      for (int i=1; i < hidDev.Inputs.RptByteLen; i++)
            s += hidDev.Inputs.DataBuf[i].ToString("X2") + " ";
```

4.6.3 ReadInput( )

public bool **ReadInput** ( ) Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Inputs provide access to read-only HID controls (buttons and values).

ReadInput uses the Win32 ReadFile( ) function to read RptByteLen bytes from the device. The endpoint used to complete this transaction is device dependent (often a non-Control endpoint).

The bytes transferred from the device are found in the Inputs.DataBuf.

NOTE: ReadInput will hang indefinitely if the device does not have an Input report ready to be read.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.ReadInput();
```

4.6.4 SetFeature( )

public bool **SetFeature**(CyUSB.CyHidValue *hidVal*, uint *val*) Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Features represent configuration data for a HID device.

SetFeature loads the Features.ReportBuf with the new configuration data and sends it to the device.

Parameters

CyHidValue hidVal
Identifies a value item of the feature to be modified.

UInt32 val
The new value to be set for this feature.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

CyHidValue hVal = hidDev.Features.Values[0];
if (hVal != null)
hidDev.SetFeature(0x20);
```

4.6.5 SetOutput( )

public System.Boolean **SetOutput** (int *rptID*) Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Outputs provide access to write-only HID controls (buttons and values) .

SetOutput uses the Win32 HidD_SetOutputReport( ) function to write RptByteLen bytes to the device using the Control endpoint.

rptID specifies the ReportID to set. The bytes to be transferred to the device should be pre-loaded into Outputs.DataBuf before calling SetOutput.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.Outputs.DataBuf[0] = hidDev.Outputs.ID;
hidDev.Outputs.DataBuf[1] = 0x01;
hidDev.Outputs.DataBuf[2] = 0x02;

hidDev.SetOutput(hidDev.Outputs.ID);
```

4.6.6 ToString( )

public override string **ToString**() Top Previous Next

Member of CyUSB.CyHidDevice

Description

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

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

string text = hidDev.ToString();
```

Fills text with the following:

```
<HID_DEVICE>
FriendlyName=""
Manufacturer="Logitech"
Product="USB-PS/2 Optical Mouse"
SerialNumber="?"
VendorID="0x046D"
ProductID="0xC03D"
Class="0x00"
SubClass="0x00"
Protocol="0x00"
BcdUSB="0x0000"
Usage="0x0002"
```

```
UsagePage="0x0001"
Version="0x2000"
<INPUT>
    RptByteLen="5"
    Buttons="1"
    Values="3"
    <BUTTON>
            Usage="0x0001"
            UsagePage="0x0009"
            UsageMax="0x0003"
            BitField="0x0002"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            IsAlias="False"
            IsRange="True"
            IsStringRange="False"
            IsDesignatorRange="False"
            IsAbsolute="True"
            StringIndex="0"
            StringMax="0"
            DesignatorIndex="0"
            DesignatorMax="0"
            DataIndex="0"
            DataIndexMax="2"
    </BUTTON>
    <VALUE>
            Usage="0x0038"
            UsagePage="0x0001"
            UsageMax="0x0038"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            IsAlias="False"
            IsRange="False"
            IsStringRange="False"
            IsDesignatorRange="False"
            IsAbsolute="False"
            HasNull="False"
            StringIndex="0"
            StringMax="0"
            DesignatorIndex="0"
            DesignatorMax="0"
            DataIndex="3"
            DataIndexMax="3"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            BitSize="8"
            ReportCount="1"
            Units="0"
```

```
            UnitsExp="0"
            LogicalMin="-127"
            LogicalMax="127"
            PhysicalMin="0"
            PhysicalMax="0"
        </VALUE>
        <VALUE>
            Usage="0x0031"
            UsagePage="0x0001"
            UsageMax="0x0031"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            IsAlias="False"
            IsRange="False"
            IsStringRange="False"
            IsDesignatorRange="False"
            IsAbsolute="False"
            HasNull="False"
            StringIndex="0"
            StringMax="0"
            DesignatorIndex="0"
            DesignatorMax="0"
            DataIndex="4"
            DataIndexMax="4"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            BitSize="8"
            ReportCount="1"
            Units="0"
            UnitsExp="0"
            LogicalMin="-127"
            LogicalMax="127"
            PhysicalMin="0"
            PhysicalMax="0"
        </VALUE>
        <VALUE>
            Usage="0x0030"
            UsagePage="0x0001"
            UsageMax="0x0030"
            BitField="0x0006"
            LinkCollection="0x0001"
            LinkUsage="0x0001"
            LinkUsagePage="0x0001"
            IsAlias="False"
            IsRange="False"
            IsStringRange="False"
            IsDesignatorRange="False"
            IsAbsolute="False"
            HasNull="False"
```

```
                StringIndex="0"
                StringMax="0"
                DesignatorIndex="0"
                DesignatorMax="0"
                DataIndex="5"
                DataIndexMax="5"
                BitField="0x0006"
                LinkCollection="0x0001"
                LinkUsage="0x0001"
                LinkUsagePage="0x0001"
                BitSize="8"
                ReportCount="1"
                Units="0"
                UnitsExp="0"
                LogicalMin="-127"
                LogicalMax="127"
                PhysicalMin="0"
                PhysicalMax="0"
            </VALUE>
        </INPUT>
        </HID_DEVICE>
```

4.6.7 WriteOutput( )

public bool **WriteOutput** ( ) Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Outputs provide access to write-only HID controls (buttons and values) .

WriteOutput uses the Win32 WriteFile( ) function to write RptByteLen bytes to the device. The endpoint used to complete this transaction is device dependent (often a non-Control endpoint).

The bytes to be transferred to the device should be pre-loaded into Outputs.DataBuf before calling WriteOutput.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.Outputs.DataBuf[1] = 0x04;
hidDev.Outputs.DataBuf[2] = 0x06;

hidDev.WriteOutput();
```

4.6.8 Capabilities

public CyUSB.HIDP_CAPS **Capabilities** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

The Capabilities property returns the HIDP_CAPS that were reported by the HID device.

4.6.9 Features

public CyUSB.CyHidReport **Features** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID Features represent configuration settings for a HID device. HID Features are Read/Write capable (use GetFeature and SetFeature).

The Features property of CyHIDDevice is a CyHidReport that embodies all the Feature items (configuration settings) reported in the HIDP_CAPS for the HID device.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidAnalyzer = HidDevices[0] as CyHidDevice;
byte[] Raw Data = new byte[hidAnalyzer.Features.RptByteLen];

if ((hidAnalyzer != null) && hidAnalyzer.GetFeature(0))
for (int x=1; x < hidAnalyzer.Features.RptByteLen; x++)
    Raw Data[x++] = hidAnalyzer.Features.DataBuf[x];
```

4.6.10 Inputs

public CyUSB.CyHidReport **Inputs** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID controls (buttons and values) are either Read-only (Inputs) or Write-only (Outputs).

Inputs is a CyHidReport that embodies all the Input controls (buttons and values) reported in the HIDP_CAPS for the HID device.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

string s = "";

if (hidDev.GetInput(hidDev.Inputs.ID))
      for (int i=1; i < hidDev.Inputs.RptByteLen; i++)
            s += hidDev.Inputs.DataBuf[i].ToString("X2") + " ";
```

4.6.11 Outputs

public CyUSB.CyHidReport **Outputs** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

HID controls (buttons and values) are either Read-only (Inputs) or Write-only (Outputs).

Outputs is a CyHidReport that embodies all the Output controls (buttons and values) reported in the HIDP_CAPS for the HID device.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

hidDev.Outputs.DataBuf[0] = hidDev.Outputs.ID;
hidDev.Outputs.DataBuf[1] = 0x01;
hidDev.Outputs.DataBuf[2] = 0x02;

hidDev.SetOutput(hidDev.Outputs.ID);
```

4.6.12 RwAccessible

public bool **RwAccessible** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

While the library is able to obtain descriptor information for all connected HID devices, Read/Write privileges are not granted, by Windows, for some HID devices.

If RwAccessible is true, the library was able to open a handle to device with Read/Write access privileges.

If RwAccessible is false, the GetXxxxx and SetXxxxx methods of the CyHidDevice class will fail.

C# Example

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

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

if (hidDev != null)
    DataXferBtn.Enabled = hidDev.RwAccessible;
else
    DataXferBtn.Enabled = true;
```

4.6.13 Tree

public override **System.Windows.Forms.TreeNode Tree** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

Tree property returns a Windows.Forms.TreeNode.

The *Text* property of the TreeNode is the Product string of the device descriptor.

The children of the node are comprised of the trees representing the HID Reports (Inputs, Outputs and Features) of the device.

The *Tag* property of the returned TreeNode contains a reference to the CyHidDevice 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, TreeView EventArgs e)
{
        TreeNode selNode = DeviceTreeView .SelectedNode;
        DescText.Text = selNode.Tag.ToString();
}
```

4.6.14 Usage

public ushort **Usage** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

Usage contains the value of the 16 bit Usage field reported in the HIDP_CAPS of the device.

4.6.15 UsagePage

public ushort **UsagePage** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

UsagePage contains the value of the 16 bit UsagePage field reported in the HIDP_CAPS of the device.

4.6.16 Version

public ushort **Version** { get; } Top Previous Next

Member of CyUSB.CyHidDevice

Description

Version contains the value of the 16 bit VersionNumber field reported in the HIDD_ATTRIBUTES of the device.

4.7 CyHidReport

public class **CyHidReport** Top Previous Next

Member of CyUSB

Description

The CyHidReport class is used by the CyHidDevice class to represent the Features, Inputs and Outputs of a HID device.

While you will never manually construct or populate a CyHidReport object, you may need to access the members of a CyHidDevice's Features, Inputs or Outputs (all of which are CyHidReport objects).

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidAnalyzer = HidDevices[0] as CyHidDevice;
byte[] Raw Data = new byte[hidAnalyzer.Features.RptByteLen];

if ((hidAnalyzer != null) && hidAnalyzer.GetFeature(0))
for (int x=1; x < hidAnalyzer.Features.RptByteLen; x++)
    Raw Data[x++] = hidAnalyzer.Features.DataBuf[x];
```

4.7.1 ID

public byte **ID** { get; } Top Previous Next

Member of CyUSB.CyHidReport

Description

ID is the report ID contained in all the items of the a given Feature, Input or Output report of a HID device.

4.7.2 NumBtnCaps

public int **NumBtnCaps** { get; } Top Previous Next

Member of CyUSB.CyHidReport

Description

NumBtnCaps returns the number of button capabilities reported for the given Feature, Input or Output of a HID device.

4.7.3 NumItems

public int **NumItems** { get; } Top Previous Next

Member of CyUSB.CyHidReport

Description

NumItems indicates the number of Value and Button items in the HID Report. NumItems is equal to NumBtnCaps + NumValues.

4.7.4 NumValCaps

public int **NumValCaps** { get; } Top Previous Next

Member of CyUSB.CyHidReport

Description

NumValCaps contains the number of value capabilities reported for the given Feature, Input or Output of a HID device.

4.7.5 NumValues

public int **NumValues** { get; } Top Previous Next

Member of CyUSB.CyHidReport

Description

NumValues represents the number of ValueCaps for the HidConstruct. If a given ValueCap is a range, NumValues is incremented by the number of values in the range.

4.7.6 RptByteLen

public int **RptByteLen** { get; } Top Previous Next

Member of CyUSB.CyHidReport

Description

RptByteLen reflects the XxxxReportByteLength field of the HIDP_CAPS structure reported for the

Features, Inputs or Outputs.

The ReportBuf for the Features, Inputs or Outputs is RptByteLen bytes long. This value includes 1 byte for the ReportID at ReportBuf[0].

4.7.7 Buttons

public CyUSB.CyHidButton[] **Buttons** Top Previous Next

Member of CyUSB.CyHidReport

Description

Buttons contains the button capabilities reported for the given Feature, Input or Output of a HID device.

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

CyHidButton hBtn = hidDev.Features.Buttons[0];
```

4.7.8 DataBuf

public byte[] **DataBuf** Top Previous Next

Member of CyUSB.CyHidReport

Description

DataBuf serves as the data buffer for the HID transfer methods. DataBuf is RptByteLen bytes long.

For most transfer operations DataBuf[0] will contain the ReportID for the transfer. Report data will begin at DataBuf[1].

C# Example

```
USBDeviceList HidDevices = new USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;
int ReportID = 0x01;

if (hidDev.GetFeature(ReportID))
{
    string s = "";
    for (int i = 1; i < hidDev.Features.RptByteLen; i++)
        s += hidDev.Features.DataBuf[i].ToString("X2") + " ";

    //FeatureDataBox.Text = s; //Apped the string to GUI data box
}
```

4.7.9 Values

public CyUSB.CyHidValue[] **Values** Top Previous Next

Member of CyUSB.CyHidReport

Description

ValueCaps contains the value capabilities reported for the given Feature, Input or Output of a HID device.

C# Example

```
USBDeviceList HidDevices = new  USBDeviceList(CyConst.DEVICES_HID);
CyHidDevice hidDev = HidDevices[0] as CyHidDevice;

CyHidValue hVal = hidDev.Features.Values[0];
if (hVal != null)
hidDev.SetFeature( 0x20);
```

4.8 CyHidValue

public class **CyHidValue** : CyUSB.CyHidButton Top Previous Next

Member of CyUSB

Description

CyHidValue represents USB Human Interface Devices Value data item. In addition to the properties of CyHidButton, It contains the following read-only properties which reflect the HID Value descriptor fields:

```
public int BitSize { get; }
public bool HasNull  { get; }
public int LogicalMax  { get; }
public int LogicalMin  { get; }
public int PhysicalMax  { get; }
public int PhysicalMin  { get; }
public uint Units { get; }
public uint UnitsExp  { get; }
```

C# Example

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

```
CyHidValue hidVal = DeviceTreeView .SelectedNode.Tag as CyHidValue;

if (hidVal != null)
bResult = curHidDev.SetOutputValue(hidVal, 5);
```

4.9 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 CyUSB.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 == 3))
        InterruptIn = ept as CyInterruptEndPoint;
```

4.10 CyIsocEndPoint

public class **CyIsocEndPoint** : CyUSB.CyUSBEndPoint

Top Previous Next

Member of CyUSB

Description

CyIsocEndPoint is a subclass of the CyUSBEndPoint abstract class. This class exists to provide special ISOC packet information handling for Isochronous transfers.

Instances of CyIsocEndPoint 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
CyIsocEndPoint IsocOut = null;

// Create a list of devices served by CyUSB.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 == 1))
        IsocOut = ept as CyIsocEndPoint;
```

4.10.1 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.

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 *singleXer* 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.

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 w ith 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++;
      }

    }
```

4.10.2 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

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.

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 w ith 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++;
      }

  }
```

4.10.3 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 w ith 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++;
    }

}
```

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

        MyDevice.IsocInEndPt.XferSize = len;

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

4.10.5 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.

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.

See also the CyIsocEndPoint.XferData method that does not pass back an array of ISO_PKT_INFO structures.

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

        MyDevice.IsocInEndPt.XferSize = len;

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

4.10.6 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.

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.

See also the CyIsocEndPoint.XferData method that passes back an array of ISO_PKT_INFO structures.

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);
      }
```

4.11 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 how you might use 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="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>
```

4.11.1 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.USBCfgs[0].ToString();
```

Fills text with the following:

```
<CONFIGURATION>
    Configuration="0"
    ConfigurationValue="1"
    Attributes="0xA0"
    Interfaces="1"
    DescriptorType="2"
    DescriptorLength="9"
    TotalLength="135"
    MaxPow er="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>
```

4.11.2 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.

4.11.3 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.

4.11.4 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.

4.11.5 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.

4.11.6 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.

4.11.7 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.

4.11.8 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.

4.11.9 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.

4.11.10 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;
```

4.11.11 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.

4.11.12 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>
```

4.12 CyUSBDevice

public class **CyUSBDevice** : CyUSB.USBDevice Top Previous Next
Member of CyUSB

Description

The CyUSBDevice class represents a USB device attached to the CyUSB.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 (ControlEndPt, BulkInEndPt, BulkOutEndPt, etc.) members.

Because CyUSBDevice is a descendant of USBDevice, it inherits all the members of USBDevice.

C# Example

```
CyControlEndPoint      CtrlEndPt           = null;

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

```
}
```

4.12.1 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 w ith address 0x82");
```

4.12.2 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);
```

4.12.3 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);
```

4.12.4 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);
```

4.12.5 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.

4.12.6 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.

4.12.7 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"
    MaxPow er="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>
```

4.12.8 UsbdStatusString( )

public string **UsbdStatusString** (uint *stat*) Top Previous Next
Member of CyUSB.CyUSBDevice

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);
}
```

4.12.9 AltIntfc

public byte **AltIntfc** { set; get; } Top Previous Next
Member of CyUSB.CyUSBDevice

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. Assignent 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
```

4.12.10 AltIntfcCount

public byte **AltIntfcCount** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

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>
```

4.12.11 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;
```

4.12.12 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.

4.12.13 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.

4.12.14 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.

4.12.15 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.

4.12.16 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.

4.12.17 DeviceHandle

public System.IntPtr **DeviceHandle** { get; }
Member of CyUSB.CyUSBDevice
Top Previous Next

Description

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

4.12.18 DriverVersion

public uint **DriverVersion** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

Description

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

4.12.19 EndPointCount

public byte **EndPointCount** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

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>
```

4.12.20 IntfcClass

public byte **IntfcClass** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

Description

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

4.12.21 IntfcProtocol

public byte **IntfcProtocol** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

Description

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

4.12.22 IntfcSubClass

public byte **IntfcSubClass** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

Description

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

4.12.23 MaxPacketSize

public byte **MaxPacketSize** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

Description

This property returns the value of the *bMaxPacketSize0* field from the open device's Device Descriptor structure.

4.12.24 MaxPower

public byte **MaxPower** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

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.

4.12.25 StrLangID

public ushort **StrLangID** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

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).

4.12.26 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();
}
```

4.12.27 USBDIVersion

public uint **USBDIVersion** { get; } Top Previous Next
Member of CyUSB.CyUSBDevice

Description

This property returns the version of the USB Host Controller Driver in BCD format.

4.12.28 BulkInEndPt

public CyUSB.CyBulkEndPoint **BulkInEndPt** Top Previous Next
Member of CyUSB.CyUSBDevice

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);
}
```

4.12.29 BulkOutEndPt

public CyUSB.CyBulkEndPoint **BulkOutEndPt** Top Previous Next
Member of CyUSB.CyUSBDevice

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);
}
```

4.12.30 ControlEndPt

public CyUSB.CyControlEndPoint **ControlEndPt** Top Previous Next
Member of CyUSB.CyUSBDevice

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.ReqType     = CyConst.REQ_VENDOR;
      CtrlEndPt.ReqCode     = 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);
    }
```

4.12.31 EndPoints

public CyUSB.CyUSBEndPoint[] **EndPoints** Top Previous Next
Member of CyUSB.CyUSBDevice

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>
```

4.12.32 InterruptInEndPt

public CyUSB.CyInterruptEndPoint **InterruptInEndPt**
Member of CyUSB.CyUSBDevice

Top Previous Next

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);
}
```

4.12.33 InterruptOutEndPt

public CyUSB.CyInterruptEndPoint **InterruptOutEndPt** Top Previous Next
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 EndPoints 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);
}
```

4.12.34 IsocInEndPt

public CyUSB.CyIsocEndPoint **IsocInEndPt** Top Previous Next
Member of CyUSB.CyUSBDevice

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);
}
```

4.12.35 IsocOutEndPt

public CyUSB.CyIsocEndPoint **IsocOutEndPt** Top Previous Next
Member of CyUSB.CyUSBDevice

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);
}
```

4.12.36 USBCfgs

public CyUSB.CyUSBConfig[] **USBCfgs** Top Previous Next
Member of CyUSB.CyUSBDevice

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).

4.13 CyUSBEndPoint

public abstract class **CyUSBEndPoint** Top Previous Next
Member of CyUSB

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 descendents) is exposed. This is because endpoint objects are automatically instantiated for you (as part of a USBDevice) when you create a USBDeviceList object.

4.13.1 Abort( )

public bool **Abort** ( ) Top Previous Next
Member of CyUSB.CyUSBEndPoint

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);
          }
       }
   }
```

4.13.2 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 *singleXer* 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);
            }
        }

    }
```

4.13.3 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.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);
            }
        }

    }
```

4.13.4 Reset( )

public bool **Reset** ( ) Top Previous Next
Member of CyUSB.CyUSBEndPoint

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.

4.13.5 ToString( )

public override string **ToString( )**() Top Previous Next
Member of CyUSB.CyUSBEndPoint

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>
```

4.13.6 WaitForXfer( )

public bool **WaitForXfer** (uint *ovlapEvent* , 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 *ovlapEvent* 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.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);
            }
        }

    }
```

4.13.7 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);
      }
```

4.13.8 Address

public byte **Address** { get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

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 CyUSB.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) );
```

4.13.9 Attributes

public byte **Attributes** { get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

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 CyUSB.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) );
```

4.13.10 bIn

public bool **bIn** { get; }
Member of CyUSB.CyUSBEndPoint

Top Previous Next

Description

bIn 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.

bIn 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 CyUSB.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.bIn)
        BulkIn2 = (CyBulkEndPoint) ept;

    e++;

} while ( (e < dev.EndPointCount) && (BulkIn2 == null) );
```

4.13.11 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.

4.13.12 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.

4.13.13 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.

4.13.14 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.

4.13.15 Interval

public byte **Interval** { get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

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.

4.13.16 MaxPktSize

public int **MaxPktSize** { get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

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) .

Example

If wMaxPacketSize is 0x1400 (binary = 0001 0100 0000 0000)

MaxPktSize = [100 0000 0000 binary] * [10 binary + 1] = 1024 * 3 = 3072

4.13.17 NtStatus

public uint **NtStatus** { get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

Description

NtStatus member contains the error code returned from the last call to the XferData or BeginDataXfer methods.

4.13.18 TimeOut

public uint **TimeOut** { set; get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

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.BulkOutEndPt != null)
      {
            int len = 512;
            byte [] buf = new  byte[len];

            MyDevice.BulkOutEndPt.TimeOut = 5000; // 5 sec time out  or set CyConst.INFINITE
            (0xFFFFFFFF) to wait forever.
            MyDevice.BulkOutEndPt.XferData(ref buf, ref len);
      }
```

4.13.19 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();
}
```

4.13.20 UsbdStatus

public uint **UsbdStatus** { get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

Description

UsbdStatus member contains an error code returned from the last call to the XferData or BeginDataXfer methods.

4.13.21 XferMode

public byte **XferMode** { set; get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

Description

The XferMode property controls how data is passed to / from the CyUSB.sys driver.

Older versions of the CyUSB.sys driver did not support transfer of data directly into or out of the user's data buffer. So, the API would create a temporary buffer to pass to the driver, then copy the user's data to/from that buffer. This double bufferring 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 bufferring
```

4.13.22 XferSize

public int **XferSize** { set; get; } Top Previous Next
Member of CyUSB.CyUSBEndPoint

Description

Each non-control endpoint has a transfer size that is some multiple of its MaxPktSize. This transfer size can be adjusted programmatically.

The transfer size establishes the size of internal buffers used by the USB driver stack for performing data transfers. Larger values for the transfer size enable data transfers involving fewer transactions. However, those larger buffers also consume more available memory.

XferSize is implemented as a property. When you assign a value to XferSize, the value is automatically rounded up to be an integral multiple of the endpoint's MaxPktSize that is greater or equal to the requested size.

Please refer given link for more information set transfer size :http://msdn.microsoft.com/en-us/library/ff538112.aspx

C# Example

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

if (MyDevice != null)
    if (MyDevice.BulkOutEndPt != null)
        MyDevice.BulkOutEndPt.XferSize = 0x4000;    // 16KB
```

4.14 CyUSBInterface

public class **CyUSBInterface** Top Previous Next
Member of CyUSB

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 how you might use 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.USBCfgs[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"
                  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>
```

4.14.1 ToString

public override string **ToString**() Top Previous Next
Member of CyUSB.CyUSBInterface

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>
```

4.14.2 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.

4.14.3 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

4.14.4 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.

4.14.5 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.

4.14.6 bInterfaceNumber

public byte **bInterfaceNumber** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

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.

4.14.7 bInterfaceProtocol

public byte **bInterfaceProtocol** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

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.

4.14.8 bInterfaceSubClass

public byte **bInterfaceSubClass** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

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.

4.14.9 bLength

public byte **bLength** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

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.)

4.14.10 bNumEndpoints

public byte **bNumEndpoints** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

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.

4.14.11 iInterface

public byte **iInterface** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

Description

This property reports the *iInterface* 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.

4.14.12 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 CyUSBInteface 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, TreeView EventArgs e)
{
        TreeNode selNode = DeviceTreeView .SelectedNode;
        DescText.Text = selNode.Tag.ToString();
}
```

4.14.13 wTotalLength

public ushort **wTotalLength** { get; } Top Previous Next
Member of CyUSB.CyUSBInterface

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.

4.14.14 EndPoints

public CyUSB.CyUSBEndPoint[] **EndPoints** Top Previous Next
Member of CyUSB.CyUSBInterface

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>
```

4.15 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 CyUSB.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;
```

4.15.1 SendScsiCmd( )

unsafe public bool **SendScsiCmd**(byte *cmd*, byte *op*
, byte *lun*, byte *dirIn*, int *bank*, int *lba*, int *bytes*,
byte[] *data*)

Top Previous Next

Member of CyUSB.CyUSBStorDevice

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);
```

4.15.2 ToString( )

public override string **ToString( )**()

Top Previous Next

Member of CyUSB.CyUSBStorDevice

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>
```

4.15.3 BlockSize

public int **BlockSize** { get; } Top Previous Next
Member of CyUSB.CyUSBStorDevice

Description

The BlockSize property reports the size of data blocks transferred by the mass storage class device.

4.15.4 TimeOut

public uint **TimeOut** { set; get; } Top Previous Next
Member of CyUSB.CyUSBStorDevice

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
```

4.16 HIDP_CAPS

public struct **HIDP_CAPS** Top Previous Next
Member of CyUSB

Description

The CyHidDevice class contains a HIDP_CAPS data member called Capabilities.

The HIDP_CAPS structure is defined as:

```
[StructLayout(LayoutKind.Sequential,Pack=1)]
public struct HIDP_CAPS
{
    public ushort   Usage;
    public ushort   UsagePage;
    public ushort   InputReportByteLength;
    public ushort   OutputReportByteLength;
    public ushort   FeatureReportByteLength;
    public ushort   R0,R1,R2,R3,R4,R5,R6,R7,R8,R9;
    public ushort   R10,R11,R12,R13,R14,R15,R16;

    public ushort   NumberLinkCollectionNodes;

    public ushort   NumberInputButtonCaps;
    public ushort   NumberInputValueCaps;
    public ushort   NumberInputDataIndices;

    public ushort   NumberOutputButtonCaps;
    public ushort   NumberOutputValueCaps;
    public ushort   NumberOutputDataIndices;

    public ushort   NumberFeatureButtonCaps;
    public ushort   NumberFeatureValueCaps;
    public ushort   NumberFeatureDataIndices;
}
```

4.17 ISO_PKT_INFO

public struct **ISO_PKT_INFO** Top Previous Next
Member of CyUSB

Description

An array of ISO_PKT_INFO structures is passed to the XferData and FinishDataXfer methods of a CyIsocEndPoint 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** Top Previous Next
Member of CyUSB

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 faciliates 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 IntPt    hEvent;
}
```

4.19 OverlapSignalAllocSize

public int **OverlapSignalAllocSize** { get; } Top Previous Next
Member of CyUSB

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** Top Previous Next
Member of CyUSB

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** Top Previous Next
(uint *lpEventAttributes*, uint *bManualReset*, uint *bInitialState*, uint *lpName*)
Member of CyUSB.PInvoke

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 w TotalLength;
        public byte bNumInterfaces;
        public byte bConfigurationValue;
        public byte iConfiguration;
        public byte bmAttributes;
        public byte MaxPow er;
}
```

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** Top Previous Next
Member of CyUSB

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 USBDevice

public abstract class **USBDevice : IDisposable** Top Previous Next
Member of CyUSB

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, CyHidDevice) 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, CyUSBStorDevice or CyHidDevice.

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 CyUSB.sys driver

// Get the FriendlyName of the first object, regardless of its class
USBDevice device = devList[0];
string fName = device.FriendlyName;
```

4.24.1 Dispose( )

public **void Dispose()** Top Previous Next
Member of CyUSB.USBDevice

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();
}
```

4.24.2 Equals( )

public override bool **Equals**(object *right*) Top Previous Next
Member of CyUSB.USBDevice

Description

The Equals method allows the comparison of two USBDevice objects (or their descendants) to determine if they reprresent 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
}
```

4.24.3 BcdUSB

public ushort **BcdUSB** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.4 DevClass

public ushort **DevClass** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.5 DevProtocol

public byte **DevProtocol** { get; } Top Previous Next
Member of CyUSB.USBDevice

Description

This property reports the value of the device descriptor's **bDeviceProtocol** field.

C# Example

```
// Create a list of devices served by the CyUSB.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;
```

4.24.6 DevSubClass

public byte **DevSubClass** { get; } Top Previous Next
Member of CyUSB.USBDevice

Description

This property reports the value of the device descriptor's **bDeviceSubClass** field.

C# Example

```
// Create a list of devices served by the CyUSB.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;
```

4.24.7 DriverName

public string **DriverName** { get; } Top Previous Next
Member of CyUSB.USBDevice

Description

DriverName returns an upper-case string that represents the USB device driver serving the USBDevice. This value will be one of the following:

CYUSB.SYS
USBSTOR.SYS
HIDUSB.SYS

C# Example

```
// Create a list of devices served by the CyUSB.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;
```

4.24.8 FriendlyName

public string **FriendlyName** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.9 Manufacturer

public string **Manufacturer** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.10 Name

public string **Name** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.11 Path

public string **Path** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.12 Product

public string **Product** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.13 ProductID

public ushort **ProductID** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.24.14 SerialNumber

public string **SerialNumber** { get; } Top Previous Next
Member of CyUSB.USBDevice

Description

SerialNumber returns the string indicated by the device descriptor's **iSerialNumber** field.

C# Example

```
// Create a list of devices served by the CyUSB.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;
```

4.24.15 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 descendents 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();
}
```

4.24.16 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 CyUSB.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;
```

4.24.17 VendorID

public ushort **VendorID** { get; } Top Previous Next
Member of CyUSB.USBDevice

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 CyUSB.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;
```

4.25 USBDeviceList

public class **USBDeviceList** : IDisposable, IEnumerable Top Previous Next
Member of CyUSB

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 CyUSB.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);
}
```

4.25.1 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
}
```

4.25.2 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
}
```

4.25.3 Dispose( )

public **void Dispose**() Top Previous Next
Member of CyUSB.USBDeviceList

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();
}
```

4.25.4 USBDeviceList( )

public **USBDeviceList** (byte *DeviceMask*) Top Previous Next
Member of CyUSB.USBDeviceList

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);
```

4.25.5 Count

public int **Count** { get; } Top Previous Next
Member of CyUSB.USBDeviceList

Description

The Count property reflects the number of USBDevice objects in the USBDeviceList.

C# Example

```
// Create a list of devices served by the CyUSB.sys driver
USBDeviceList usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
if (usbDevices.Count == 0) return;
```

4.25.6 USBDeviceList [int index]

public const CyUSB.USBDevice **this** [int *index*] Top Previous Next
Member of CyUSB.USBDeviceList

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, a CyUSBStorDevice or a CyHidDevice to be of much use.

C# Example

```
// Create a list of devices served by the CyUSB.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;
}
```

4.25.7 USBDeviceList [string fName]

public const CyUSB.USBDevice **this** [string *FriendlyName*] Top Previous Next
Member of CyUSB.USBDeviceList

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, a CyUSBStorDevice or a CyHidDevice to be of much use.

C# Example

```
// Create a list of devices served by the CyUSB.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;
}
```

4.25.8 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, a CyUSBStorDevice or a CyHidDevice to be of much use.

C# Example

```
// Create a list of devices served by the CyUSB.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;
}
```

4.25.9 USBDeviceList [int VID, int PID, int UsagePg, int Usage]

public const CyUSB.USBDevice **this** [int *VendorID*, int *ProductID*, int *UsagePage*, int *Usage*]
Member of CyUSB.USBDeviceList

Top Previous Next

Description

This index operator provides access to elements of the USBDeviceList based on the VendorID, ProductID, UsagePage and Usage properties of the USBDevice objects in the list.

Note that UsagePage and Usage are properties of the CyHidDevice class. So, only CyHidDevice objects in the USBDeviceList will have a chance of being accessed using this index operator.

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.

int *UsagePage*

UsagePage will be compared to the *UsagePage* property of the devices in the list in order to locate a particular device.

int *Usage*

Usage will be compared to the *Usage* property of the devices in the list in order to locate a particular device.

Return Value

Returns the first USBDevice object that matches all of the *VendorID*, *ProductID*, *UsagePage*, and *Usage*. Because USBDevice is an abstract class, the object returned will need to be casted into a CyHidDevice to be of much use.

C# Example

```
// Create a list of devices served by the HID driver
```

```
USBDeviceList hidDevices = new USBDeviceList(CyConst.DEVICES_HID);
if (hidDevices.Count == 0) return;

// Get the first device having VendorID == 0x046D, ProductID == 0xC03D,
// UsagePage == 1, and Usage == 2
CyHidDevice mouse = hidDevices[0x046D, 0xC03D, 1, 2] as CyHidDevice ;

if (mouse != null)
{
      CyHidReport inputs = mouse.Inputs;
}
```

4.25.10 USBDeviceList [string sMfg, string sProd]

public const CyUSB.CyHidDevice **this** [int VID, int PID]
Member of CyUSB.USBDeviceList

Top Previous Next

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, a CyUSBStorDevice or a CyHidDevice to be of much use.

C# Example

```
// Create a list of devices served by the CyUSB.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;
```

4.25.11 USBDeviceList [string sMfg, string sProd, int UsagePg, int Usage]

public const CyUSB.CyHidDevice **this** [string *Manufacturer*, string *Product*, int *UsagePage*, int *Usage*]
Member of CyUSB.USBDeviceList

Top Previous Next

Description

This index operator provides access to elements of the USBDeviceList based on the Manufacturer, Product, UsagePage and Usage properties of the USBDevice objects in the list.

Note that UsagePage and Usage are properties of the CyHidDevice class. So, only CyHidDevice objects in the USBDeviceList will have a chance of being accessed using this index operator.

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.

int *UsagePage*

UsagePage will be compared to the *UsagePage* property of the devices in the list in order to locate a particular device.

int *Usage*

Usage will be compared to the *Usage* property of the devices in the list in order to locate a particular device.

Return Value

Returns the first USBDevice object that matches all of the *Manufacturer*, *Product*, *UsagePage*, and *Usage*. Because USBDevice is an abstract class, the object returned will need to be casted into a CyHidDevice to be of much use.

C# Example

```
// Create a list of devices served by the HID driver
USBDeviceList hidDevices = new USBDeviceList(CyConst.DEVICES_HID);
if (hidDevices.Count == 0) return;

// Get the first device having Manufacturer == "Cypress", Product == "WirelessUSB",
// UsagePage == 0xff01, and Usage == 1
CyHidDevice wusb_Battery = hidDevices["Cypress","WirelessUSB",0xff01,1] as CyHidDevice;

if (wusb_Battery != null)
{
```

```
        CyHidReport features = w usb_Battery.Features;
    }
```

4.26 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.

4.26.1 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 withing 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 FwOffset     = 0;

        ArrayList codeLines = new ArrayList();

        TgtDevice.FillCodeList(FwFile,codeLines);

        bool bParsed = Util.ParseHexData(codeLines,FwImage, ref ImageLen, ref FwOffset);
}
```

4.26.2 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(FwFile, FwImage, ref ImageLen, ref FwOffset);
}
```

4.26.3 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 FwOffset       = 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[] FwImage= new  byte[Util.MaxFwSize];
      Util.ParseIICData(FwBuf, FwImage, ref ImageLen, ref FwOffset);

      ResetFX2(1); // Halt

      ushort chunk = 2048;
      byte [] buffer = new  byte[chunk];

      CyControlEndPoint ep0 = CyDevice.ControlEndPt;

      for (ushort i=FwOffset; i<ImageLen; i+=chunk)
      {
            ep0.Value = i;
            int len = ((i + chunk)<ImageLen) ? chunk : ImageLen - i;
            Array.Copy(FwImage,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.ReqType       = CyConst.REQ_VENDOR;
        ep0.Value               = 0xE600;
        ep0.Index               = 0x0000;

        ep0.ReqCode       = 0xA0;
        dta[0]               = hold;
        int len              = 1;

        ep0.Write(ref dta, ref len);

        //Thread.Sleep(500); //Wait for some time
    }
```

4.26.4 ParseIICFile( )

Top Previous Next

public static bool **ParseIICFile** (System.String *fName* , byte[] *FwBuf* , ref ushort *FwLen* , ref ushort *FwOff*)
Member of CyUSB.Util

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(FwFile, FwImage, ref ImageLen, ref FwOffset);
    }
```

4.26.5 ReverseBytes( )

public static int **ReverseBytes** (byte* *dta* , int *bytes*)
Member of CyUSB.Util

Top Previous Next

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.

4.26.6 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.

4.26.7 Assemblies

public static string **Assemblies** { get; }

Member of CyUSB.Util

Top Previous Next

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\tVERSION\r\n\n
Assembly1\t\t\tAssembly1Version\r\n
Assembly2\t\t\tAssembly2Version\r\n
AssemblyN\t\t\tAssemblyNVersion\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);
}
```

4.26.8 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).

4.27 XMODE

public enum **XMODE** Top Previous

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.

Older versions of the CyUSB.sys driver did not support transfer of data directly into or out of the user's data buffer. So, the API would create a temporary buffer to pass to the driver, then copy the user's data to/from that buffer. This double bufferring 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;
```

Index

- D -

- E -

- U -