Application Note AN14557

Developing USB Applications

Author: Amit Nanda / Greg Nalder
Associated Project: N/A
Software Version: SuiteUSB.net 2.0
Associated Application Notes: None

Abstract

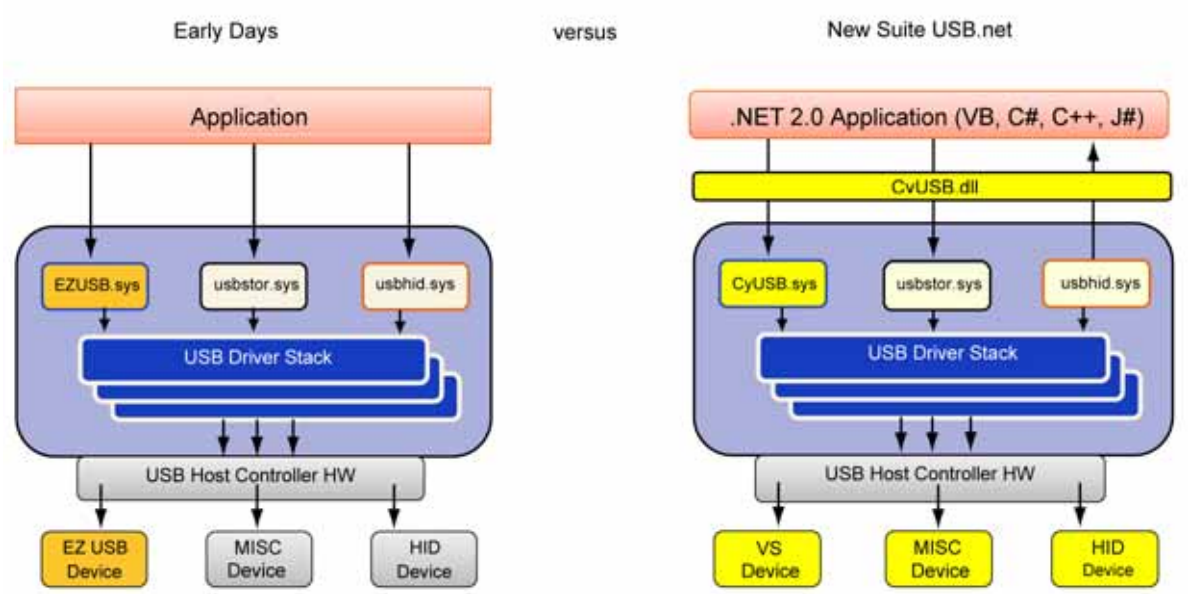
Developing USB applications has changed dramatically from the early days of USB development. Cypress has developed tools to help simplify the design of these applications. The latest addition to the family is SuiteUSB.net 2.0, a .NET application development library. The focus of this article includes a history and walk-through of writing your first USB application.

Introduction

Applications' communication with USB devices have evolved dramatically. In the early days, the application writing process involved making direct calls to drivers, something that was easier said than done. The process of writing an application was cumbersome; the application had to first get a device handle and then call device IO controls or read/write files. This made access, especially to mass storage class devices, very difficult.

In that initial development, Cypress released USB Developers' µStudio, which provided an easier, high level programming interface through which there was no need to get device handles. The API was implemented as a statically linked library. But it was limited in that it only accessed devices bound to the *cyusb.sys* driver.

The new generation of application development tools from Cypress has an even simpler, more powerful API. SuiteUSB.net 2.0 supports the *cyusb.sys*, *usbstor.sys*, and *usbhid.sys* device drivers increasing the spectrum of devices that can be accessed with this tool. The figure below illustrates the early days of application development versus the new Suite USB.net.



Cypress' new SuiteUSB.net enables users to quickly develop custom USB applications. At the heart of the new SuiteUSB.net is c*yusb.dll*. This DLL is a managed Microsoft .NET 2.0 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 name space has to include a line to include the name space in the appropriate syntax.

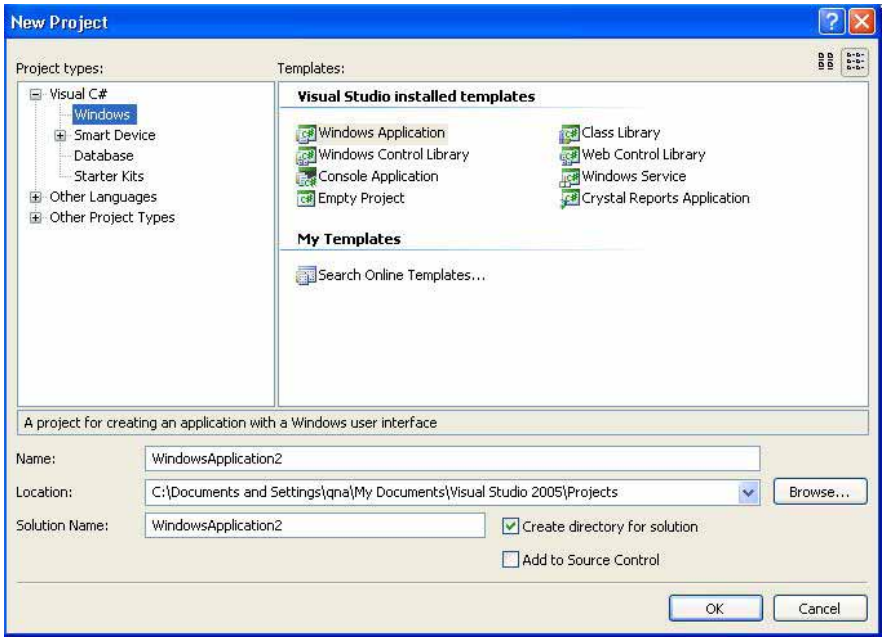
The following walks you through developing your first application with SuiteUSB. The examples shown below are written in C# but can be converted to C++ or J#.

Ease of *Cyusb.dll*

Cyusb.dll is a major leap forward from the previous development tools offered by Cypress. It manages USB devices for you, so there are no more 'open' and 'close' of devices. You can locate multiple USB devices connected to the host using various indexers into the USBDeviceList. You can easily populate a USBDeviceList with devices attached to more than one driver (including HID and MSC devices). *Cyusb.dll* easily handles USB PNP events.

Writing Your First Application

Before you begin writing your first application, ensure you have installed *SuiteUSB.net*, then start Visual Studio 2005 and choose Templates > New Project > Windows Application. Make sure you give your application a unique name. In this example, the application name is 'WindowsApplication2'.



Click **OK** and a blank form displays. This form is a functional application; click on the green arrow to start the application. There are several interesting things you can do with *CyUSB.dll*. To use this library, you need to add a reference to *CyUSB.dll* to your project's References folder. Any source file that accesses the CyUSB name space must include a line to include the name space in the appropriate syntax.

1. Right click on Reference and **Add Reference**, under the Solution Explorer window. Browse to the installation directory of SuiteUSB and double-click *CyUSB.dll*. This references the library to your project; however, you cannot use it just yet.
2. If you have the blank form showing, right click outside in the white space and click **View Code**. This is your code view window. Notice that you have some initial code that Microsoft puts in to start your project.
3. At the top, you can see directives, starting with the word 'Using'. For C/C++ users, these are the same as `'#include'`. Since you added the reference to *CyUSB.dll*, you have to inform the application that it is going to be used. So, under the last 'using' line, add the words: **using CyUSB**; this gives you access to the library's APIs, classes, and other functionality.
4. Actually, the 'using' directive serves as a shortcut in the code so that you do not have to explicitly reference the CyUSB name space when you use its functions. You can leave out the 'using' directive if you pre-pended 'CyUSB' to all the CyUSB member references.
5. Insert the code below into your application; it is explained later in this application note.

```
1      USBDeviceList usbDevices;
2      CyUSBDevice myDev;
3         public Form1()
4         {
5            InitializeComponent();
6
7            App_PnP_Callback evHandler = new App_PnP_Callback(PnP_Event_Handler);
8
9            usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB, evHandler);
10
11           // Get the first device having VendorID == 0x04B4 and ProductID == 0x8613
12
13           myDev = usbDevices[0x04B4, 0x8613] as CyUSBDevice;
14
15        }
16
17        public void PnP_Event_Handler(IntPtr pnpEvent, IntPtr hRemovedDevice)
18        {
19            if (pnpEvent.Equals(CyConst.DBT_DEVICEREMOVECOMPLETE))
20            {
21                usbDevices.Remove(hRemovedDevice);
22                // Other removal event handling
23            }
24
25            if (pnpEvent.Equals(CyConst.DBT_DEVICEARRIVAL))
26            {
27                usbDevices.Add();
28                // Other arrival event handling
29
30            }
31        }
```

Application Code Analysis

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. The 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, such as (line 9):

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

Selector masks are defined as follows:

1. CyConst.DEVICES_CYUSB – mask to select devices that are bound to the CyUSB driver.
2. CyConst.DEVICES_HID – mask to select devices that are in the HID class.
3. CyConst.DEVICES_MSC – mask to select devices that are in the MSC class.

These USBDevice objects have all been properly initialized and are ready for use. Once an instance of the USBDeviceList class is constructed, the USBDeviceList index operators make it easy to locate a particular device and begin using it, such as (line 13) :

`CyUSBDevice myDev = usbDevices[0x04B4, 0x8613] as CyUSBDevice;.`

If you want to search for other methods of indexing through the device list, type: **CyUSBDevice myDev = usbDevices[**. Once you type in the open bracket, Visual Studio displays different methods of using the USBDeviceList index. Because USBDeviceList implements the IEnumerable interface, you iterate through a USBDeviceList object's items using the 'foreach' keyword.

The App_PnP_Callback class is used to setup Windows PnP event handling for the devices in a USBDeviceList. To enable handling of PnP events, an instance of App_PnP_Callback is passed to the USBDeviceList constructor; `usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB, evHandler);.` The 'evHandler' is the instantiation of the class App_PnP_Callback. To create an instance of App_PnP_Callback, the name of an event handler method must be passed to the constructor; in this case, '`PnP_Event_Handler`'. The event handler method receives two parameters when the PnP event occurs. The first pnpEvent indicates what event caused the invocation of the handler. The second hRemovedDevice is a handle to the device that is being removed in the case of device removal events.

The PnP_Event_Handler funciton takes two arguments, as mentioned above. The function, as written, checks to see if the event was cause by a device being plugged in, '`pnpEvent.Equals(CyConst.DBT_DEVICEARRIVAL)`' or a device being removed, '`(pnpEvent.Equals`

`(CyConst.DBT_DEVICEREMOVECOMPLETE).'` You can add other functionality to this function if other work needs to be done when a device is plugged in or removed.

You can write something in the PnP_Event_Handler and then test the code.

1. Inside the IF statement that handles device removal, type the following: **Text = "Device Yanked"**.
2. Inside the IF statement that handles device arrival, type the following: **Text = "Device Arrived"**.

   The 'Text' property controls the text seen in the top left hand corner when you run your application. For example, if you run the application without the above mentioned code, the word 'Form1' is displayed – not very informative information. By adding this code every time a device is plugged in or removed, the software displays the text provided.
3. Press the green **Play** button and attach and detach a USB device.

   Make sure it is a Cypress USB device, because the event handler you just wrote only handles devices tied to the CyUSB driver, for now.
4. Unplug and plug the device repeatedly and watch the text change. That is your first application.

So far, only the basics of writing your first application have been covered. The next few sections discuss some advance features that are used to make the application more productive.

Advanced Topics

Before proceeding to advanced topics, a more detailed discussion about *CyUSB.dll* is required. One of the most important classes in the library is CyUSBDevice. The CyUSBDevice class represents a USB device attached to the *CyUSB.sys* device driver. A list of CyUSBDevice objects are 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, and others) members. Because CyUSBDevice is a descendant of USBDevice, it inherits all the members of USBDevice.

CyUSBDevice provides three main components (an in depth list is located in the *SuiteUSB Programmers Reference Guide*).

1. Functions (in C# parlance, these are called methods)
2. Properties
3. Objects

These three components give you access to most of the USB controls you need in your application, including functions such as GetDeviceDescriptor() and Reset(); properties such as AltIntfc and ConfigCount; and objects such as BulkIn/Out Endpt and IsocEndpt.

The first application you wrote allowed you to detect plug and play events and change the text of the application. You can add some buttons to your form and experiment with some alternate interfaces. The next example uses the EZ-USB FX2LP™. Download the *CyStream.iic* file onto the EEPROM of the EZ-USB FX2LP.

When you finish downloading the file, reset the device and make sure the Windows Device Manager detects it as a CyStream device.

Add Buttons and Toggle a 7-Segment Display

1. Click on the tab in Visual Studio that reads Form1.cs[Design].
2. Click on View > Toolbox.
3. Once the Toolbox opens, click and drag **Button** anywhere on your application.
4. A button labeled Button1 appears on your form. Double-click the **Button**.
5. Windows just created an event handler for you. Anytime a user clicks the button, your program does what is inside this function call.

   ```
   private void button1_Click(object sender,
   EventArgs e)

   object sender – where the event came
   from. If you have multiple functions
   calling this function you can determine
   where it came from.

   EventArgs e – any arguments that are
   passed in when the event happens.
   ```

6. Add code to control the 7-segment display. In your function, type the following:

   ```
   myDevice.AltIntfc++;
   Text = myDevice.AltIntfc.ToString();
   ```

   What does the above code do? As was discussed before, AltIntfc is a property that is used to get or set the alternate interface setting for the device. Since the 7-segment display is an alternate interface, this code increments the numbers on the display. At the same time, the text in your application outputs what is currently displayed. You can use this code in an application, such as a thermometer to display the temperature of a device on screen if the device is hidden inside a box.
7. Run your application. Then implement a button to decrement the count.

Detecting Devices

The following code generates an application that detects all devices connected to the bus.

1. Move the buttons to one side of your form.
2. Drag and drop **TreeView** onto the form and expand it to take up most of the room on the form.
3. Right click in the white area outside of your form and click **View Code**.

4. Add code to the function: Form1( )

```
foreach (USBDevice dev in usbDevices)
treeView1.Nodes.Add(dev.Tree);
```

After adding just two lines of code, the program gets all devices connected to the host and fills in the tree. Every USB device has a Tree property associated with it. When you call *dev.Tree*, the configuration of all the devices is returned and displayed.

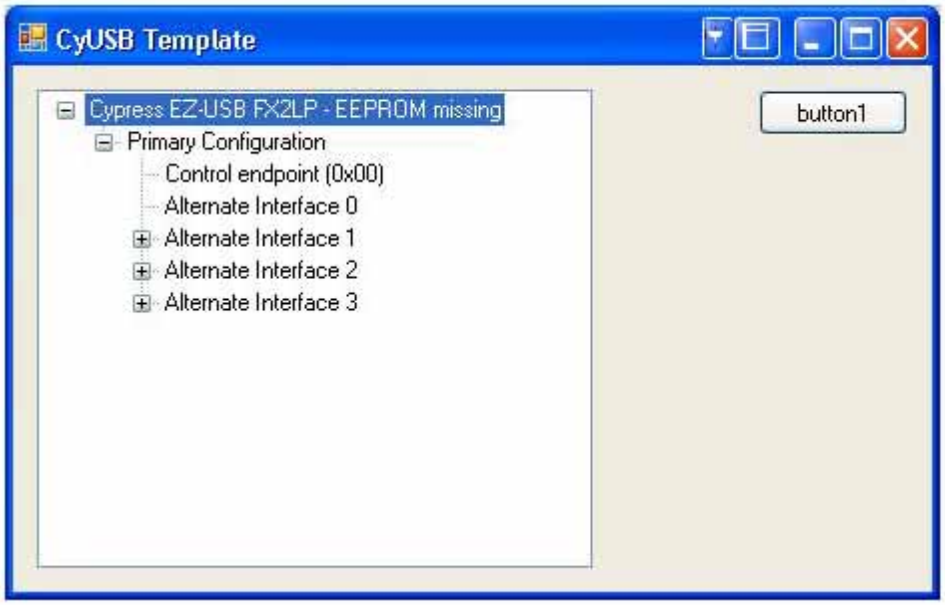
5. Add the following code to the PnP function handler. But before doing so, you have to add one more line to ensure you do not get repeat devices showing up in the tree. Add the following code under each of the two If statements in the PnP handler:

```
treeView1.Nodes.Clear();

foreach (USBDevice dev in usbDevices)
treeView1.Nodes.Add(dev.Tree);
```

This clears the tree every time a device is plugged or unplugged and then it re-initializes the tree.

Your view should look similar to the one below.



Once you perform the previous exercises, try writing and testing your own applications. Experiment with different properties and tools and see if you can write an application that meets your requirements. Reference the online Programmers Guide included in the SuiteUSB installation.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com