Cypress USBSuite Application Development

Quick Start Guide

Version 1.2.1

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intl): 408.943.2600
http://www.cypress.com

Contents

1 Introduction

1.1 Overview

Application communication with USB devices have evolved. Earlier, the application writing process was complex and involved making direct calls to drivers. The application had to first get a device handle and then call device I/O controls or read/write files.

Cypress released CyAPI.lib, first in its USB Developers' µStudio and then in the latest Cypress USBSuite. This provided a high level programming interface to get a device handle and communicate with Cypress USB devices.

The new generation of application development tools from Cypress has a simpler, more powerful API. This includes a Cypress USBSuite C++ library, CyAPI.lib and a Cypress USBSuite C# library, CyUSB.dll.

CyAPI.lib provides a simple, powerful C++ programming interface to USB devices. More specifically, it is a C++ class library that provides a high-level programming interface to the CyUsb3.sys kernel mode driver. The library is only able to communicate with USB devices that are served by (i.e. matched to) this driver.

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.

Figure 1 illustrate the earlier application development environment.

Figure 2 offers a comparison of the development environment between the C++ and the C# libraries.

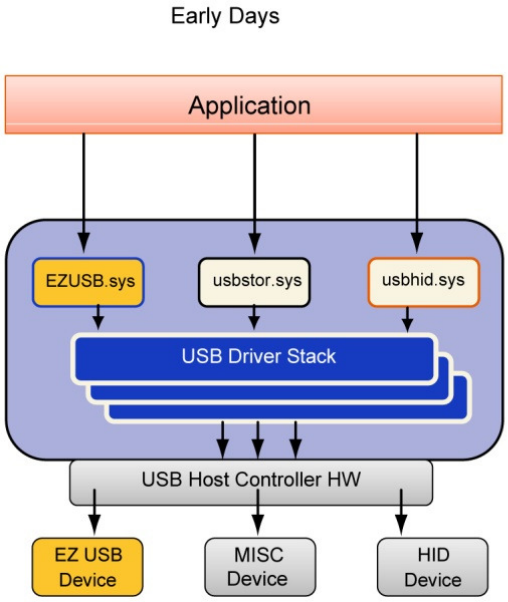

Figure 2: Early Days of Application Development

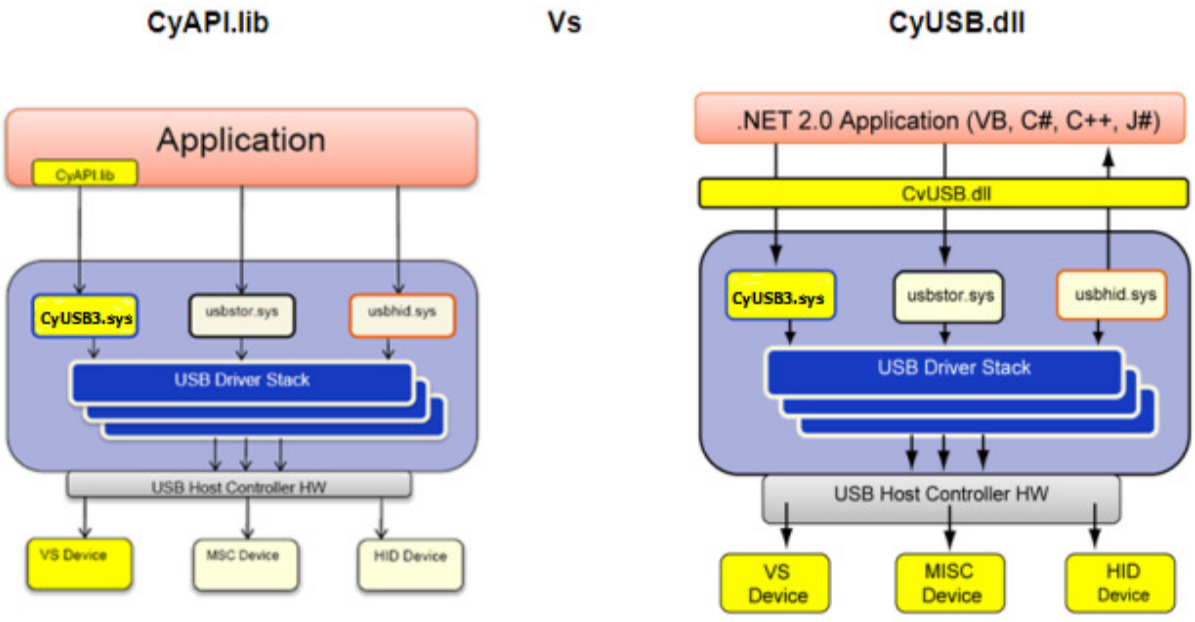

Figure 3: C++ vs C#

1.2 Cypress USBSuite

Cypress provides USBSuite, which is a set of development tools for Visual Studio to create .NET Windows applications which includes the following:

- A Generic USB Kernel Mode Driver
- A .NET Managed Class Library that has
  - CyUSB.dll, which is a C# library
  - CyAPI.lib, which is a C++ library
- USB Control Center: that serves as a USB experimenter's work-bench
- Sample Code

Note that Cypress USBSuite and Cypress USB kernel mode driver (CyUSB3.sys) are compatible only with XP, Vista, and Windows 7 and Windows 8 beta.

2 C++ Library CyAPI.lib

2.1 Overview

CyAPI.lib is implemented as a statically linked library. It provides C++ programming interface to USB devices and enables users to quickly develop custom USB applications. It enables users to access devices bound only to the cyusb3.sys driver. CyAPI.lib automatically takes care of activities such as error handling, which were otherwise handled by the user when making direct calls to the drivers.

As CyAPI.lib is a statically linked C++ library, its classes and functions can be accessed from C++ compilers such as Microsoft Visual C++. To use the library, you need to add a reference to CyAPI.lib to your project's Source Files folder and include a dependency to the header file CyAPI.h. Then, any source file that accesses the CyAPI.lib has to include a line to include header file CyAPI.h in the appropriate syntax.

The following section explains how you can develop your first application with CyAPI.lib. The following examples are written in Visual C++.

2.2 Writing Your First Application

Before you begin writing your first application, ensure you have installed USBSuite and Microsoft Visual Studio 2008. The following steps guide you on how to develop your first VC++ application using CyAPI.lib.

1. Start a new project in Visual Studio 2008 by clicking on **File > New > Project**.
2. In the window, select **Visual C++ > Windows Forms Application**, and give your application a unique name. In this example, the application name is ‘Example1’ as shown in Figure 4.

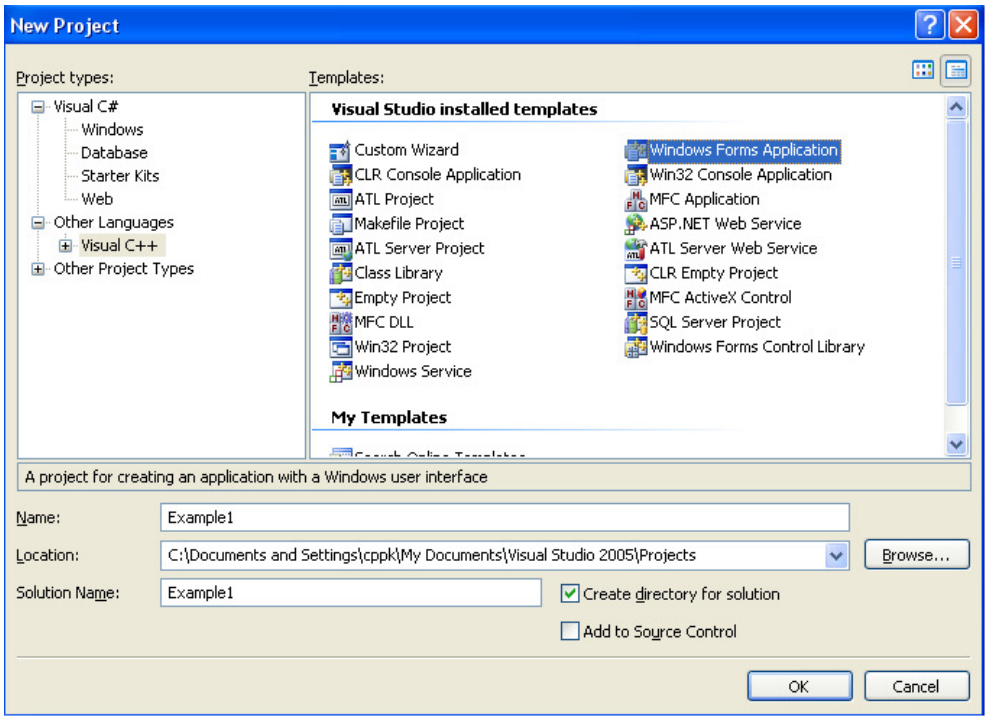

Figure 4. Starting a New Project

3. Click OK and a blank form displays. This form is a functional application. Click the green arrow (Run button) to start the application. A blank form appears as you start the application.
4. Right click on Source Files and select Add > Existing Item, under the Solution Explorer window as shown in Figure 5. Browse to the installation directory of USB Suite and choose the CyAPI.lib according to the system you are using, and double click on CyAPI.lib. This references the library to your project. However, you cannot use it just yet.

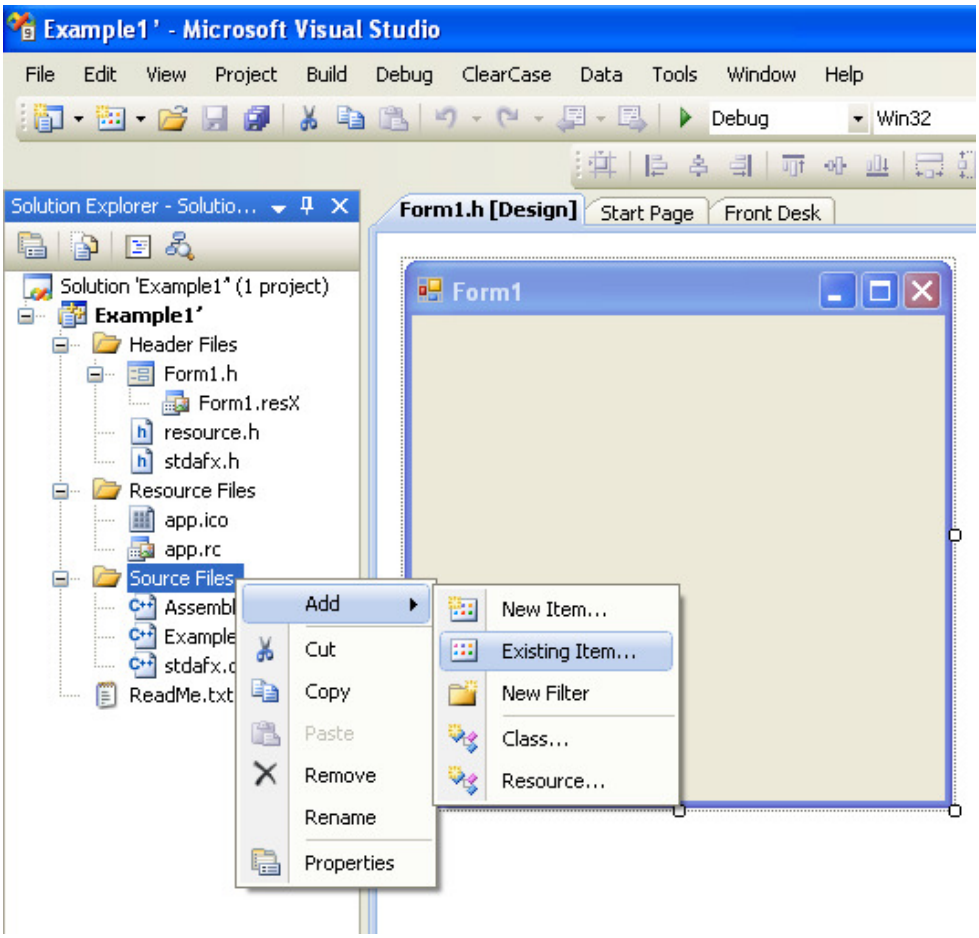

Figure 5. Adding Reference to CyAPI.lib

5. If a blank form displays, right click in the white space and click **View Code**. This is your code view window. Note that Microsoft enters some initial code to start your project.

6. At the top, you see a number of 'using Namespace' directives. Include the lines

```
#include <wtypes.h>
#include <dbt.h>
```

after the line #pragma. These two headers are required for the primitive datatypes in CyAPI.h and the USB Plug and Play (PnP) events respectively.

If a blank form displays, right click in the white space and click View Code. This is your code view window. Note that Microsoft enters some initial code to start your project

7. After adding the reference to CyAPI.lib, you have to expose the interface to it. This can be done by including a line to referenc CyAPI.h, which gives you access to the library's APIs, classes, and other functionality. This is done in the following steps:

a. Go to **Project > Properties**. In the dialog box, select **Configuration Properties > C/C++ > General > AdditionalInclude Directories**. Point it to the inc folder which contains the CyAPI.h file. Click **OK**.

b. Add a line

```
#include "CyAPI.h"
```

after the lines added in step 6

8. Go to **Project > Properties**. In the dialog box, select **Configuration Properties > Linker > Input > Additional Dependencies** and type **user32.lib** as shown in Figure 6.

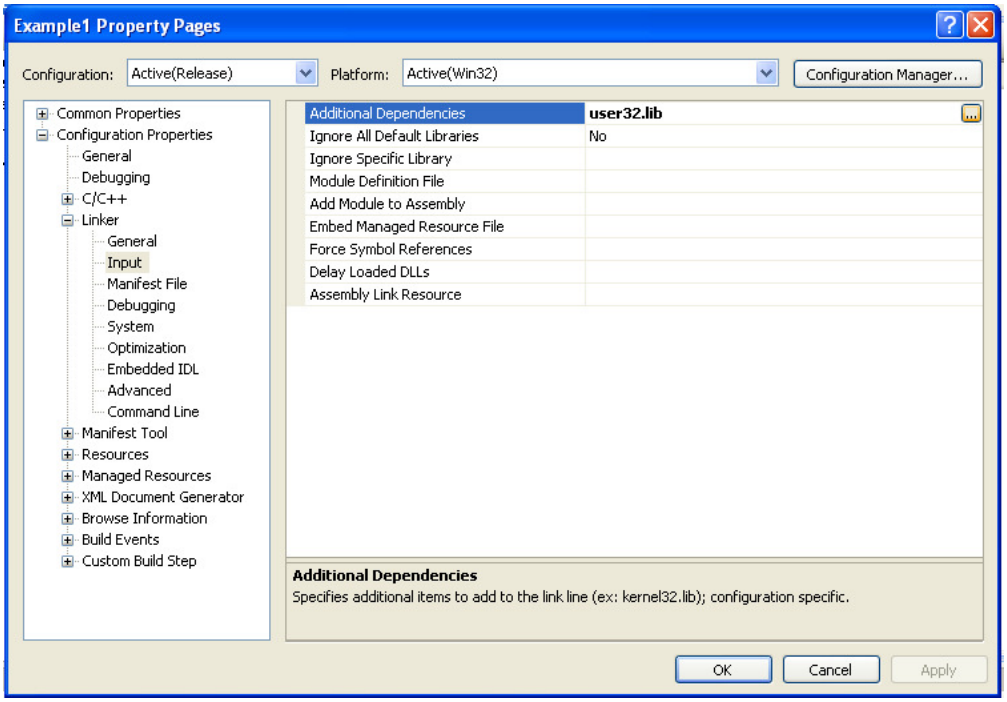

Figure 6. Additional Project Settings

9. Add the CyAPI.h header files to your project from the Cypress USBSuite installation directory CyAPI\inc. Note that the other related header files are available in the same directory.

10. Go to **Project > Properties**. In the dialog box, select **Configuration Properties > General > Common Language Runtime Support** and set it to **Common Language Runtime Support** (/clr) as shown in Figure 7.

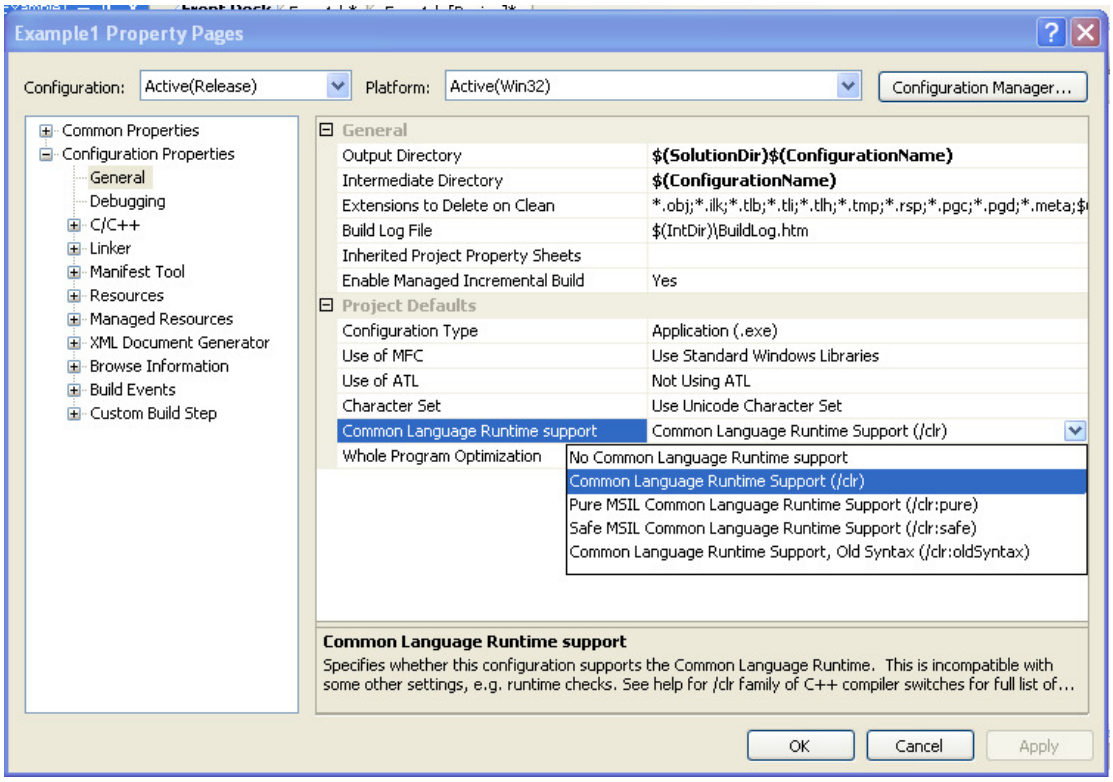

Figure 7. Project Property Settings

11. Insert the following code in your application at the exact location in the Form1 class. The code is explained in the section, Application Code Analysis. Note that Form1() and WndProc must be public members.

```
1. public ref class Form1 : public
   System::Windows::Forms::Form
2. {
3. public:
4. CCyUSBDevice *USBDevice, *CyStreamdev;
5. int AltInterface;
6. bool bPnP_Arrival;
7. bool bPnP_Removal;
8. bool bPnP_DevNodeChange;
9. Form1(void)
10.     {
11.       InitializeComponent();
12.       USBDevice =new CCyUSBDevice((HANDLE)this-
   >Handle,CYUSBDRV_GUID,true);
13.     }
14.     virtual void WndProc( Message% m ) override
```

```
15.     {
16.       if (m.Msg == WM_DEVICECHANGE)
17.       {
18.     // Tracks DBT_DEVNODES_CHANGED followed by
  DBT_DEVICEREMOVECOMPLETE
19.         if (m.WParam == (IntPtr)DBT_DEVNODES_CHANGED)
20.         {
21.               bPnP_DevNodeChange = true;
22.               bPnP_Removal = false;
23.         }
24.     // Tracks DBT_DEVICEARRIVAL followed by
  DBT_DEVNODES_CHANGED
25.         if (m.WParam == (IntPtr)DBT_DEVICEARRIVAL)
26.         {
27.               bPnP_Arrival = true;
28.               bPnP_DevNodeChange = false;
29.         }
30.         if (m.WParam == (IntPtr)DBT_DEVICEREMOVECOMPLETE)
31.               bPnP_Removal = true;
32.     // If DBT_DEVICEARRIVAL followed by
  DBT_DEVNODES_CHANGED
33.         if (bPnP_DevNodeChange && bPnP_Removal)
34.         {
35.          bPnP_Removal = false;
36.               bPnP_DevNodeChange = false;
37.               GetDevice();
38.         }
39.     // If DBT_DEVICEARRIVAL followed by
  DBT_DEVNODES_CHANGED
40.        if (bPnP_DevNodeChange && bPnP_Arrival)
41.         {
42.               bPnP_Arrival = false;
43.               bPnP_DevNodeChange = false;
44.               GetDevice();
45.         }
46.       }
47.        Form::WndProc( m );
48.     }
49.     void GetDevice()
50.     {
51.        USBDevice = new CCyUSBDevice((HANDLE)this-
  >Handle,CYUSBDRV_GUID,true);
52.        AltInterface = 0;
53.        if (USBDevice->DeviceCount())
54.        {
55.           Text = "Device Attached";
```

```
56.         }
57.         else
58.         {
59.             Text = "No Devices Attached";
60.         }
61.     }
```

2.3 Application Code Analysis

Before we analyse the previous code, note that you can find more details about the CyAPI.lib APIs in the API guide (CyAPI.chm or CyAPI.pdf).

An application normally creates an instance of the CCyUSBDevice class that knows how many USB devices are attached to the **CyUsb3.sys** driver. Therefore, a working knowledge of the CCyUSBDevice class is essential. The CCyUSBDevice class is the primary entry point in the library. All the functionality of the library should be accessed through an instance of CCyUSBDevice. An instance of CCyUSBDevice is aware of all the USB devices that are attached to the CyUSB3.sys driver and can selectively communicate with any one of them by using the Open ( ) method. The CCyUSBDevice object created serves as the programming interface to the driver whose GUID is passed in the guid parameter. The constructor of this class is displayed in line number 12:

```
USBDevice = new CCyUSBDevice((HANDLE)this-
>Handle,CYUSBDRV_GUID,true);
```

(HANDLE)this->Handle is a handle to the application's main window (the window whose WndProc function processes USB PnP events).

Pass CYUSBDRV_GUID as the guid parameter. CYUSBDRV_GUID is a unique constant guid value for the CyUSB3.sys driver and is specified in the inf file that is used to bind the device to the CyUSB3.sys driver.

These CCyUSBDevice objects are all properly initialized and ready to use.

MainForm's WndProc method is used to watch for PnP messages. Windows sends all top-level windows a set of default messages when new devices or media are added and become available, and when existing devices or media are removed. These messages are known as WM_DEVICECHANGE messages. Each of these messages has an associated event, which describes the change.

When a device is added or removed from the system, the system broadcasts the DBT_DEVNODES_CHANGED device event using the WM_DEVICECHANGE message. The operating system sends the DBT_DEVICEARRIVAL device message when a device is inserted and becomes available. Similarly, a _DEVICEREMOVAL device message is sent when a device is removed.

The WndProc takes the message sent by the operating system as an argument and if the message indicates a device arrival or a device removed status, calls the GetDevice() function to update the status of USB devices connected to the host bound to the CyUSB3.sys driver.

The GetDevice() function uses the DeviceCount() function (line number 53), which is a member of the CCyUSBDevice class. The DeviceCount() function returns the number of devices attached to the CyUSB3.sys driver. If this function returns a non-zero value, it means that there is one, or more, devices connected to the host that is bound to the CyUSB3.sys driver. You can write an IF statement as shown in the previous example (line number 53 to 60) to check for the presence or absence of USB devices.

1. Inside the IF statement that indicates that there are devices attached, type the following:

   Text = “Device Attached”.
2. Inside the else statement that indicates that no devices are attached, type the following:

   Text = "No Devices Attached".

These lines of code display the status of device connection to host. If there are one or more devices connected to host, the “Device Attached” text is displayed. If no devices are attached, then the “No Devices Attached” text is displayed. The **Text** property controls the text seen at the top left corner when you run your application. For example, if you run the application without the code, the word **Form1** is displayed, which is not very informative. Adding this code every time a device is plugged in or removed updates the text.

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 write only handles devices tied to the CyUSB driver.

4. Unplug and plug the device repeatedly and watch the text change.

These are the basics of writing your own application. The next few sections discuss features that make the application more productive.

2.4 Additional Features in the Application

The CCyUSBDevice provides the following two components (a detailed list is located in the CyAPI Programmers Reference Guide):

Functions

Properties (Data Members)

These two components give you access to most of the USB controls needed for your application, including functions such as GetDeviceDescriptor(), Reset() and SetAltIntfc(); properties such as DeviceName, DevClass, VendorID (VID), and ProductID (PID).

Detecting Devices

The first application you wrote allowed you to detect PnP events and change the text of the application. This section explains how you can create a Listview that displays the currently connected USB devices. The following code generates an application that detects all devices connected to the bus.

1. Click **Form1.h [Design]** tab in Visual Studio.
2. Click **View > Toolbox**.
3. Drag and drop listBox in the form and expand it to take up most of the room on the form.
4. Right click in the white space outside of your form and click **View Code**.
5. Insert the following code to Form1.

```
1. void RefreshList()
2. {
3.    listBox1->Items->Clear();
4.    listBox1->Text = " ";
5.    for(int i=0;i<USBDevice->DeviceCount();i++)
6.        {
7.           USBDevice->Open(i);
8.           listBox1->Items->Add(gcnew String(USBDevice-
   >FriendlyName));
9.           listBox1->Text= Convert::ToString(listBox1-
   >Items[i]);
10.       }
11.     }
```

This function connects all the devices to the CyUSB3.sys driver and displays their names in a listbox. The DeviceCount() function is implemented in CyAPI.lib and returns the number of USB devices attached to the host, which are bound to CyUSB3.sys driver. This function should be called every time a device is attached or removed to update the list of devices. This single function fills the **listBox** with the friendly names of all the USB devices bound to the CyUSB3.sys driver.

listBox1→ Items→Clear(); – It clears the tree every time the function is called. The open() function gives a handle to ith USBdevice attached to the CyUSB3.sys driver and the **FriendlyName** property contains the device

description string for the open device, which was provided by the driver's .inf file.

Add the following line of code

```
RefreshList();
```

This code calls the above function in GetDevice() and inside Form() constructor. When the application starts, the Listbox is populated with the initial list of devices attached. When a USB PnP event occurs (attach/detatch), the listbox gets populated with a fresh list of currently connected USB devices.

3 C# Library CyUSB.dll

3.1 Overview

Cypress's new USBSuite.net enables users to quickly develop custom USB applications. At the heart is cyusb.dll. This DLL is a managed Microsoft .NET class library. It provides a high level, powerful programming interface to USB devices. Instead of communicating with USB device drivers directly through Win32 API calls (such as SetupDiXxxx and DeviceIoControl), applications can access USB devices through 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 must add a reference to cyusb.dll to your project's References folder. Then, any source file that accesses the CyUSB name space must include a line to add the name space in the appropriate syntax.

3.2 Writing Your First Application

Before you begin writing your first application, ensure you have installed USBSuite and Visual Studio 2008. The following steps guide you to develop your first VC# application using CyUSB.dll.

1. Start Visual Studio 2008 and choose File > New Project > Windows Form Application.
2. In the window that pops up, make sure you give your application a unique name. In this example, the application name is 'WindowsFormApplication1' as shown in Figure 8.

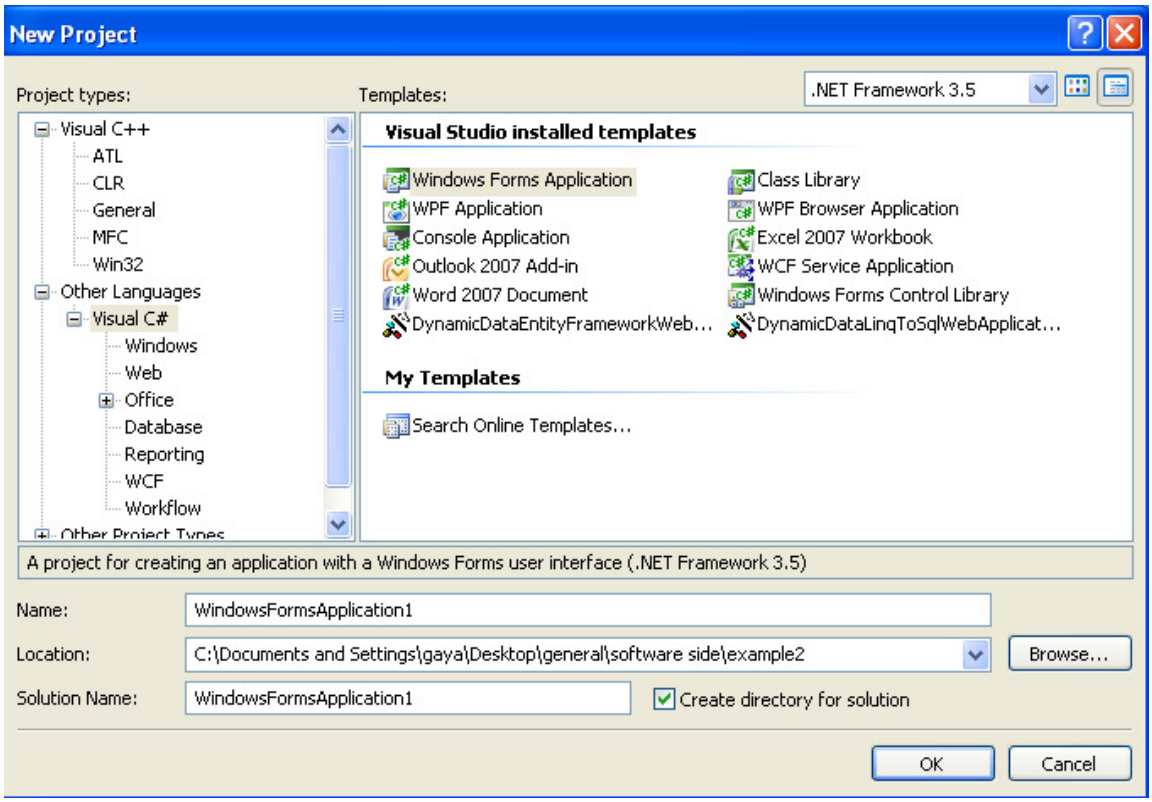

Figure 8. First Application - WindowsFormApplication1

3. Click **OK** and a blank form displays. This form is a functional application. Click on the green arrow (Run button) to start the application. The blank form appears as you start the application.
4. There are many interesting things you can do with CyUSB.dll. To use this library, you must 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.
5. For that, right click on **Reference** and **Add Reference**, under the Solution Explorer window as in Figure 9. Select the Browse tab from the window that pops up and browse to the installation directory of USBSuite and double-click CyUSB.dll. This references the library to your project. However, you cannot use it just yet.

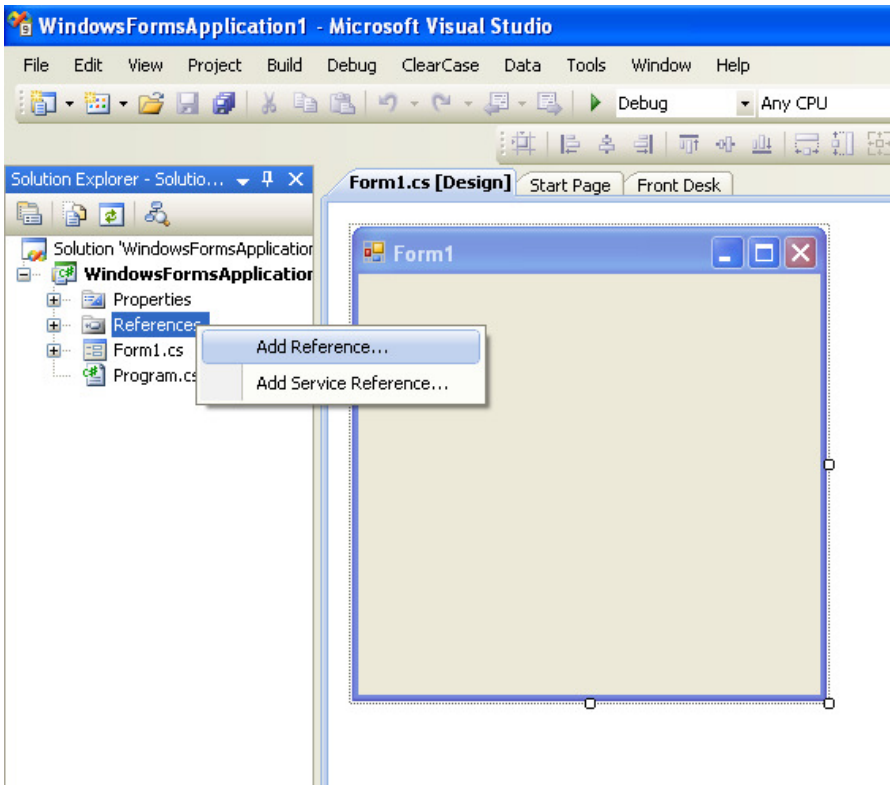

Figure 9. Adding reference to CyUSB.dll

6. If you see the blank form, right click outside in the white space and click View Code. This is your code view window. Note that Microsoft enters initial code to start your project.
7. At the top, there are 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 must inform the application that it is going to be used. Under the last 'using' line, add the words: using CyUSB; this gives you access to the library's APIs, classes, and other functionality.
8. 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 affix 'CyUSB' to all the CyUSB member references.
9. Insert the following code in your application (refer to Application Code Analysis).

```
1.   USBDeviceList usbDevices;
2.   CyUSBDevice myDevice;
3.   public Form1()
4.   {
5.      InitializeComponent();
```

```
6.      usbDevices = new
   USBDeviceList(CyConst.DEVICES_CYUSB);
7.      usbDevices.DeviceAttached +=new
   EventHandler(usbDevices_DeviceAttached);
8.      usbDevices.DeviceRemoved += new
   EventHandler(usbDevices_DeviceRemoved);
9.
10.       // Get the first device having VendorID == 0x04B4
   and ProductID == 0x1003
11.        myDevice = usbDevices[0x04B4, 0x1003] as
   CyUSBDevice;
12.    }
13.    void usbDevices_DeviceAttached(object sender,
   EventArgs e)
14.    {
15.       //Add code to handle Device arrival
16.    }
17.    void usbDevices_DeviceRemoved(object sender,
   EventArgs e)
18.    {
19.       //Add code to handle Device removal
20.    }
```

3.3 Application Code Analysis

Before we analyse the previous code, note that you can find more details about the CyUSB.dll APIs in the API guide (CyUSB.NET.chm or CyUSB.NET.pdf).

An application normally creates an instance of the USBDeviceList class, which represents a list of USB devices. Thus a good working knowledge of the USBDeviceList class is essential. The USBDeviceList represents a dynamic list of USB devices that are accessible through 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 6) in the previous code snippet:

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

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 are all properly initialized and are ready for use. After 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 11) in the previous code snippet:

```
CyUSBDevice myDevice = usbDevices[0x04B4, 0x1003] as
CyUSBDevice;
```

If you want to search for other methods of indexing through the device list, type: **CyUSBDevice myDev = usbDevices[.** After 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.

Windows PlugNPlay (PnP) event handling for the devices in a USBDeviceList are also supported by the library. To enable handling of PnP events,USBDeviceList provides two event handlers DeviceAttached and DeviceRemoved.

DeviceAttached - When a new USB device is plugged into the bus, the arrival of a new USB device is detected by this event. Handling of the event requires that an EventHandler object be assigned to the DeviceAttached event handler.

```
usbDevices.DeviceAttached+=new
EventHandler(usbDevices_DeviceAttached);
```

The previous line of code assigns `usbDevices_DeviceAttached` as the event handler for the DeviceAttached Event. Any action to be taken on arrival of the USB device should be done within the following event handler function

```
void usbDevices_DeviceAttached(object sender, EventArgs e)
```

DeviceRemoved - When a USB device is disconnected from the bus, the removal of the USB device is detected by this event. Handling of the event requires that an EventHandler object be assigned to the DeviceRemoved event handler.

```
usbDevices.DeviceRemoved += new
EventHandler(usbDevices_DeviceRemoved);
```

The above line of code assigns `usbDevices_ DeviceRemoved` as the event handler for the DeviceRemoved Event. Any action to be taken on removal of the USB device should be done within the following event handler function:

```
void usbDevices_DeviceRemoved(object sender, EventArgs e)
```

You can write something in both the Event Handlers and then test the code.

1. Inside the `usbDevices_DeviceRemoved` event handler, type the following:**Text = "Device Removed";**

2. Inside the `usbDevices_DeviceAttached` event handler, type the following: **Text = "Device Attached";**

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 handlers 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.

The next few sections discuss features used to make the application more productive.

3.4 Additional Features in the Application

Before proceeding to the next topic, 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 CyUSB3.sys device driver. A list of CyUSBDevice objects are generated by passing DEVICES_CYUSB mask to the USBDeviceList constructor. After you obtain a CyUSBDevice object, you can communicate with the device through 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 USBSuite 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.

3.4.1 Detecting Devices

This section explains how to create a tree view that displays the currently connected USB 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 when the application starts. 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 both the PnP event handlers. Before doing this, you have to add one more line to ensure repeat devices do not show up in the tree, that is, to clear the treeview first and then populate it with the fresh list of devices. Add the following code under both `usbDevices_DeviceAttached` and `usbDevices_DeviceRemoved` event handlers:

```
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 figure shown below .

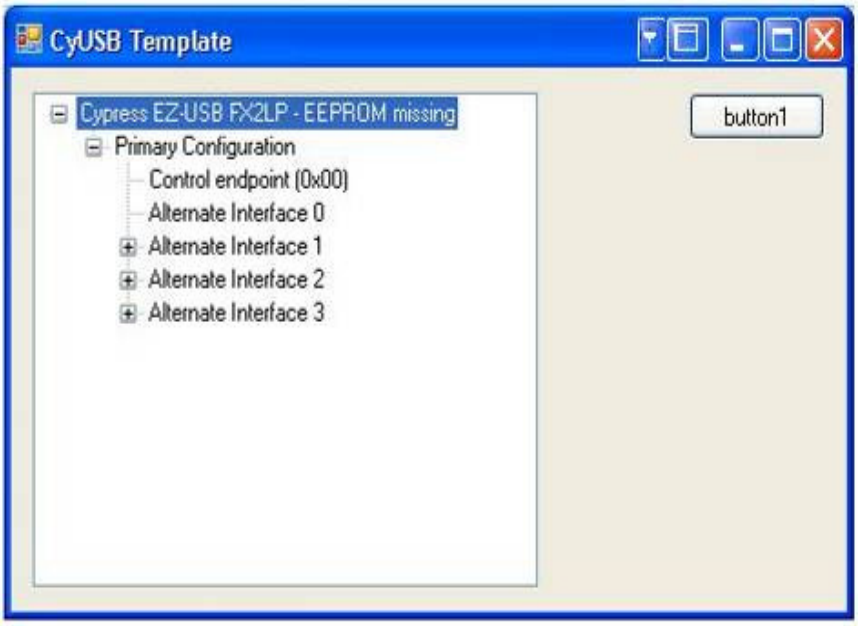

Figure 9 CYUSB Template