Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 001-61744

Spec Title: INTRODUCTION TO CYAPI.LIB BASED APPLICATION DEVELOPMENT USING VC++ - AN61744

Sunset Owner: Nikhil Naik (NIKL)

Replaced By: NONE

AN61744

Introduction to CyAPI.lib Based Application Development Using VC++

Author: Praveen Kumar CP
Associated Project: No
Associated Part Family: CY7C68013/ CY7C68013A
Software Version: Microsoft Visual Studio 2008/ 2010

OBSOLETE

Abstract

SuiteUSB 3.4 is a .NET application development library from Cypress to create Windows Applications using Microsoft Visual Studio. AN61744 includes a history and guides you on how to write your first USB application on Microsoft Visual C++ platform using Cypress Suite USB C++ library, CyAPI.lib. This document is primarily meant for beginners in Windows Application and CyAPI.lib

Contents

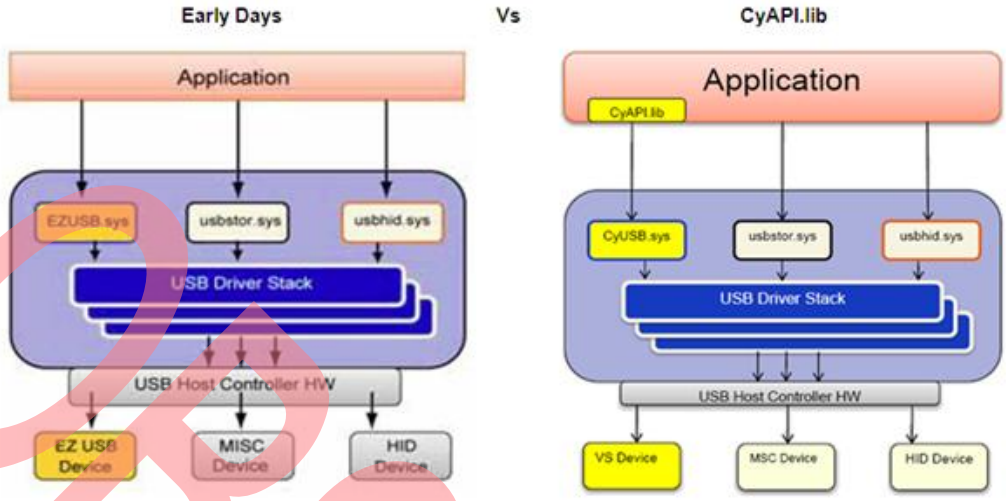
Introduction 1
Cypress SuiteUSB 2
Cyusb.dll versus CyAPI.lib 2
System Requirements 3
Hardware 3
Software 3
Writing Your First Application 3
Application Code Analysis 7
Additional Features in the Application 8
Detecting Devices 8
Add Buttons and Toggle a 7-Segment Display 9
Summary 11
Advanced Examples 11
Additional Resources 11
About the Author 11
Appendix 12
Document History 15
Worldwide Sales and Design Support 16

Introduction

Applications' communication with USB devices have evolved. Earlier, the application writing process involved making direct calls to drivers. The process of writing an application was complex; the application had to first get a device handle and then call device I/O controls, or read/write files. This made it difficult to access USB devices.

Cypress released CyAPI.lib, first in its USB Developers' µStudio and then in the latest Cypress SuiteUSB. This provided a high level programming interface to get a device handle and communicate with Cypress USB devices. Figure 1 on page 2 compares the early days of application development with the CyAPI.lib

Figure 1. Early Days of Application Development Vs. the CyAPI.lib



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. *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 *cyusb.sys* driver. Instead of communicating with USB device drivers directly through Win32 API calls, such as SetupDi and DeviceIoControl, applications can access USB devices through library functions such as XferData and properties such as AltIntfc.

Because 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++.

Cypress SuiteUSB

The host application is developed using Cypress SuiteUSB C++ library (CyAPI.lib) that comes with Cypress SuiteUSB. Cypress provides SuiteUSB, which is a set of development tools for Visual Studio to create .NET Windows applications. SuiteUSB.NET 3.4 includes the following:

- A Generic USB Device 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 Codes.

For more details, go to SuiteUSB 3.4

Note Cypress Suite USB and Cypress USB driver CyUSB.sys are compatible only with Windows 2000, XP, Vista, and Windows 7. If you need USB applications on MAC or Linux, you can refer to the following examples. These examples show you how to communicate with Cypress USB devices using the generic open source driver LIBUSB:

- MAC Users: EZ-USB® FX2LP™ - Developing USB Application on MAC OS X using LIBUSB
- Linux Users: EZ-USB® FX2LP™/ FX3™ Developing Bulk-Loop Example on Linux

Cyusb.dll versus CyAPI.lib

CyUSB.dll is a managed Microsoft .NET class library Because it manages USB devices for you, there are no more 'open' and 'close' of devices. You can locate multiple USB devices connected to the host using various indexers in the USBDeviceList. It provides simpler and more powerful APIs. CyUSB.dll supports the *cyusb.sys*, *usbstor.sys*, and *usbhid.sys* device drivers increasing the spectrum of devices that can be accessed with this tool. *Cyusb.dll* easily handles USB PNP events too. For more information about CyUSB.dll, refer to the files *CyUSB.NET.chm* or *CyUSB.NET.pdf*, located at `C:\Cypress\Cypress Suite USB 3.4.x\CyUSB.NET` after installing SuiteUSB 3.4. If you want to develop your application using CyUSB.dll, you should refer to Introduction to CyUSB.dll based Application development using C#, which also guides you to more practical examples in that.

OBSOLETE

System Requirements

Hardware

A FX2LP DVK (CY3684) board is used as the development and testing platforms for this example. A detailed schematic of the DVK can be found at the location `C:\Cypress\USB\Hardware\FX2LP` after installing FX2LP DVK. More information about the board is available in the 'EZ-USB Advanced Development Board' section of the 'EZ-USB_GettingStarted' document, available at `C:\Cypress\USB\doc\General` (after you install DVK).

Software

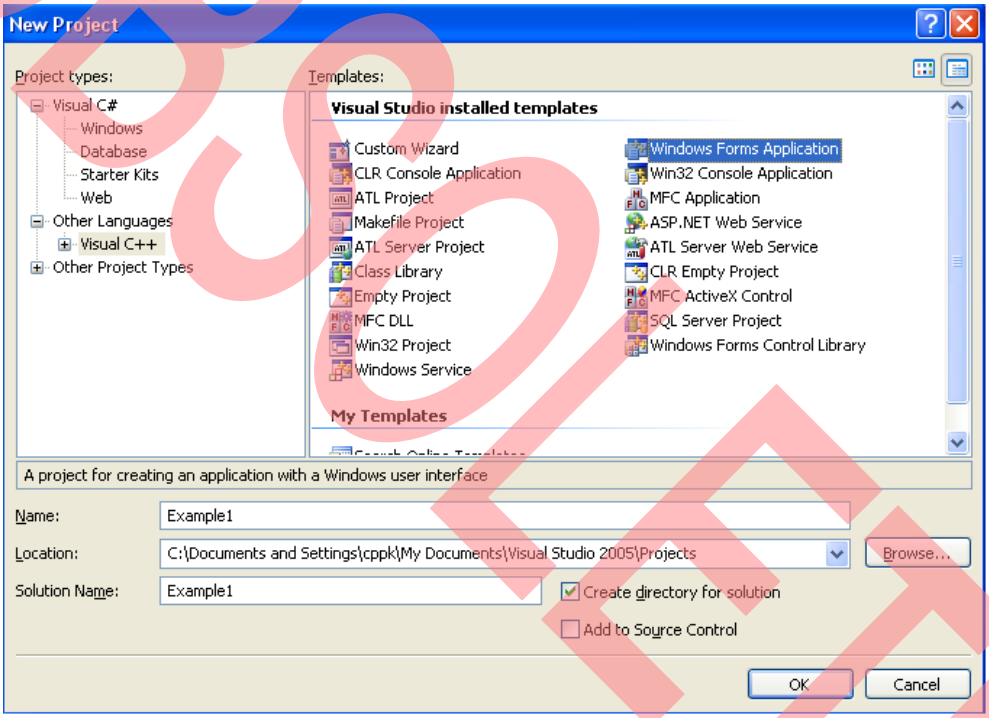
1. Cypress SuiteUSB 3.4
2. Microsoft Visual Studio 2008.

Writing Your First Application

Before you begin writing your first application, ensure you have installed SuiteUSB 3.4 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 **Other Languages** > **Visual C++** > **Windows Forms Application**, and give your application a unique name. In this example, the application name is 'Example1' as shown in Figure 2.

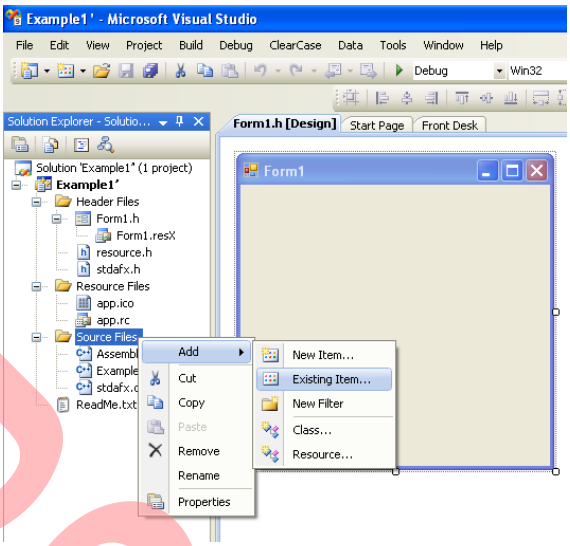
Figure 2. Starting a New Project



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 3 on page 4. Browse to the installation directory of SuiteUSB 3.4 (`C:\Cypress\Cypress Suite USB 3.4.6\CyAPI\lib`) 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.

OBSOLETE

Figure 3. 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.

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 that is found in the following path

```
C:\Cypress\Cypress Suite USB
3.4.x\CyAPI\inc
```

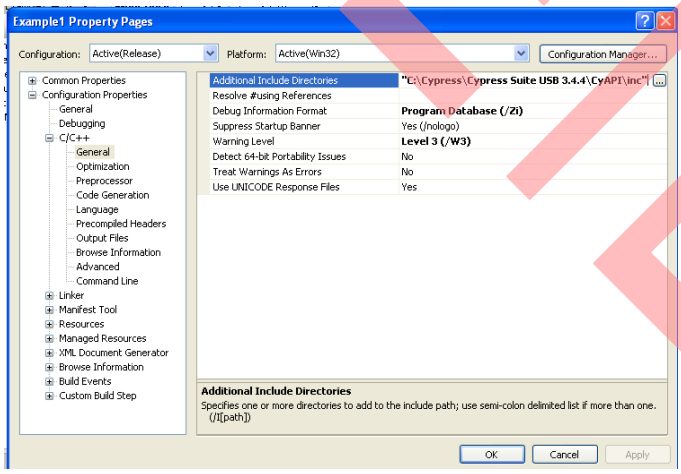
   after the installation of SuiteUSB 3.4 as illustrated in Figure 4. This folder contains the *CyAPI.h* file. Click **OK.**

   b. Add a line

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

   after the lines added in step 6.

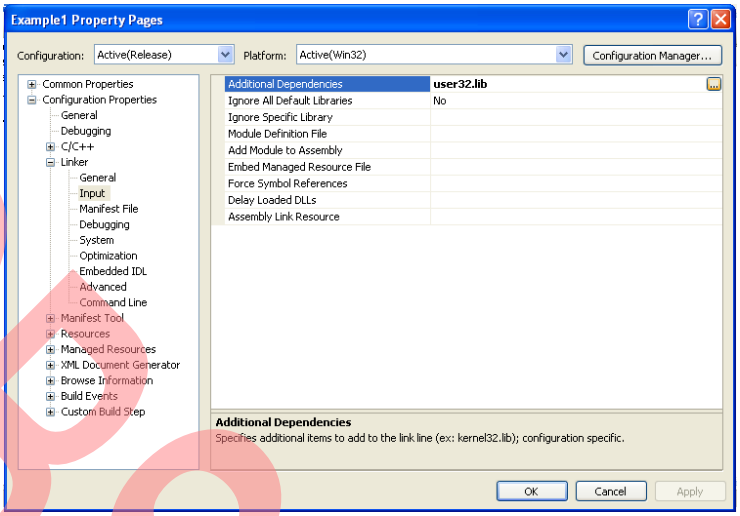
Figure 4. Settings to include the CyAPI.lib Path



OBSOLETE

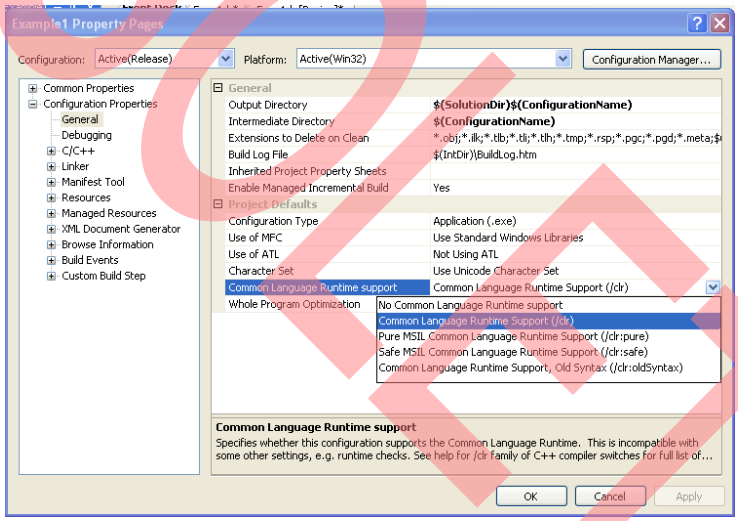
8. Go to **Project** > **Properties**. In the dialog box, select **Configuration Properties** > **Linker** > **Input** > **Additional Dependencies** and type **user32.lib** as shown in Figure 5.

Figure 5. Additional Project Settings



9. 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 6.

Figure 6. Project Property Settings



10. 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 on page 7. 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);
```

OBSOLETE

```
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. }
```

OBSOLETE

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*), which you can find at `C:\Cypress\Cypress Suite USB 3.4.x\CyAPI` after installing SuiteUSB 3.4.

An application normally creates an instance of the CCyUSBDevice class that knows how many USB devices are attached to the **CyUsb.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 CyUSB.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 *CyUSB.sys* driver and is specified in the *inf* file that is used to bind the device to the *CyUSB.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 *CyUSB.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 *CyUSB.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 *CyUSB.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.

OBSOLETE

Additional Features in the Application

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

1. Functions
2. 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 CyUSB.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 *CyUSB.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 *CyUSB.sys* driver.

listBox1→ Items→Clear(); – It clears the tree every time the function is called. The open() function gives a handle to i^{th} USBdevice attached to the *CyUSB.sys* driver and the **FriendlyName** property contains the device description string for the open device, which was provided by the driver's *.inf* file.

6. 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. Your view should be similar to Figure 7. For exact positioning of the code statements in various functions, refer to the Appendix section on page 12.

Figure 7. Application GUI



OBSOLETE

Add Buttons and Toggle a 7-Segment Display

Now, add some buttons to your form and experiment with alternate interfaces. The next example uses the EZ-USB® FX2LP™.

1. This example, uses a specific fimrware to demonstrate advanced features in the CyAPI.lib.dll. Use the CyStream firmware, which is located at `C:\Cypress\Cypress Suite USB 3.4.x\Firmware\CyStreamer` after installing SuiteUSB 3.4.
2. Now, download the *CyStream.iic* file to the EEPROM of the EZ-USB FX2LP.
3. To do that, first connect the FX2LP DVK with the "EEPROM ENABLE" switch to 'No EEPROM' position to the PC.
4. It enumerates with the default internal descriptor. Use the appropriate CyUSB.inf file to bind with the device. For more information on binding the driver, see the section, MatchingDriverToUSBDevice, in *CyUSB.chm* at `C:\Cypress\Cypress Suite USB 3.4.x\Driver` after installing SuiteUSB 3.4 or the link Drivers for FX1/FX2LP to bind with the device.
5. Change the 'EEPROM ENABLE' switch position to "EEPROM" position, and the EEPROM SELECT switch to the LARGE EEPROM position on the device.
6. Open the Control Center application present at **Start** > **Programs** > **Cypress** > **Cypress Suite USB 3.4.x** > **Control Center.** Download the CyStream.iic from `C:\Cypress\Cypress Suite USB 3.4.x\Firmware\CyStreamer` to the large EEPROM present on the FX2LP DVK using the Control Centre utility.
7. After you reset the FX2LP, the device enumerates running the CyStream firmware.
8. If Windows pops up asking you to bind the driver, repeat the steps explained in Step 4.
9. The CyStream example has a number of Alternate settings for interface. The selected Alternate setting is displayed on the 7-segment display on the FX2LP DVK. For more information on the CyStream example, refer to the source code CyStream.uV2, available at `C:\Cypress\Cypress Suite USB 3.4.x\Firmware\CyStreamer`.
10. Now, add an event handler on your application to select the Alternate setting of the CyStream device.

Follow these steps to modify your application:

1. In the Toolbox, click and drag the Button anywhere on your application.
2. A button labeled Button1 appears on your form. Double click the Button.
3. An event handler is created. When you click the button, your program does what is inside this function call.

   ```
   private:System::Void
   button1_Click(System::Object^sender,
   System::EventArgs^  e)
   ```

   object^ sender – where the event came from. If there are multiple functions, calling this function can determine where it came from.

   EventArgs^ e – any arguments that are passed in when the event happens.
4. Add the following code to select the Alternate setting of the CyStream device. In your function, type the following:

   ```
   CyStreamdev->SetAltIntfc
   (++AltInterface);

   Text = Convert::ToString (CyStreamdev-
   >AltIntfc());
   ```

   Note that you need to declare and define "CyStreamdev" before you can use that inside this event handler. That is explained in step 5.

   TheSetAltIntfc (UCHAR alt) function is used to set the alternate interface setting for the device to the value alt. Thus, when you click on Button1, you set the next alternate interface setting on CyStream device. Since the 7-segment display shows the alternate interface, this code increments the numbers on display. At the same time, the text in your application outputs what is currently displayed. You can use this code in an application where a USB device has multiple alternate interfaces, each operating with its own set of endpoints. The interface currently used can be displayed on the screen.

OBSOLETE

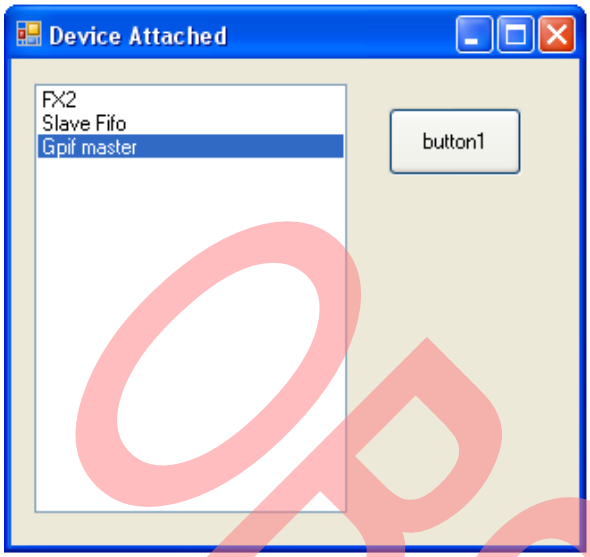
5. You already declared CyStreamdev, in class Form1. (Refer to Line 4 in the code snippets in Writing Your First Application). CyStreamdev is instantiated in the modified RefreshList() function as follows:

```
1.        void RefreshList()
2.       {       int vid, pid;
3.
4.               listBox1->Items->Clear();
5.               listBox1->Text = " ";
6.
7.                CyStreamdev = NULL;
8.                button1->Enabled = FALSE;
9.               for(int i=0;i<USBDevice->DeviceCount();i++)
10.         {
11.            USBDevice->Open(i);
12.            listBox1->Items->Add(gcnew String(USBDevice->
                     FriendlyName));
13.            listBox1->Text= Convert::ToString(listBox1->Items[i]);
14.             vid = USBDevice -> VendorID;
15.             pid = USBDevice -> ProductID;
16.             if(vid == 0x04B4 && pid == 0x1003)
17.             {
18.                CyStreamdev = new CCyUSBDevice(
                        (HANDLE)this->Handle,CYUSBDRV_GUID,true);
19.                button1 ->Enabled = TRUE;
20.             }
21.
22.          }
23.         }
```

6. In Line 7 of the previous code snippet, CyStreamdev is assigned NULL by default. While you populate the Listbox with all the devices attached, you must also check the VID/PID of the particular device (lines 14,15 of the code snippet). If the VID/PID equals 0x04B4/0x1003 (same as that for CyStream firmware), then you instantiate the "CyStreamdev" with a handle to that particular device and enable Button1. Note that Button1 is disabled by default (line 8 in the code snippet).
7. Now run your application. Your application looks similar to Figure 8. Then try to implement a button to decrement the count.

OBSOLETE

Figure 8. Application GUI



After you perform these exercises, try to write and test your own applications. Experiment with different properties and tools to write an application that meets your requirements. Refer to the CyAPI Programmers Reference Guide included in the SuiteUSB installation.

Summary

This application note explained how to write a simple application on VC++ using CyAPI.lib. You can use this application as a stepping stone to develop real world applications involving data transfer to and from Cypress devices. Refer to the sections, Advanced Examples and Additional Resources, for more details.

Advanced Examples

Now that you know how to write a simple application on VC++ using CyAPI.lib, you may want to know how to develop more practical applications to transfer data to and from the device. More application examples are provided along with Cypress SuiteUSB. Refer to the examples at `C:\Cypress\CypressSuiteUSB3.4.4\CyAPI\examples` after installing SuiteUSB.

Additional Resources

- Getting Started with FX2LP™
- Introduction to CyUSB.dll based Application development using C# - Helps in getting started with developing host applications in VC# using CyUSB.dll.
- EZ-USB® FX2LP™ Host Application in VC++ 2008 Using Suite USB Library (CyUSB.dll) – Advanced example on developing applications on VC++ using CyUSB.dll
- EZ-USB FX2LP™ Bulk Transfer Application in C# Using SuiteUSB C# Library (CyUSB.dll) – Advanced example on developing applications on VC# using CyUSB.dll

About the Author

Name:	Praveen Kumar C P
Title:	Application Engineer

OBSOLETE

Appendix

```
#pragma once
#include <wtypes.h>
#include <dbt.h>
#include "CyAPI.h"
namespace Example1 {
      using namespace System;
      using namespace System::ComponentModel;
      using namespace System::Collections;
      using namespace System::Windows::Forms;
      using namespace System::Data;
      using namespace System::Drawing;

      /// <summary>
      /// Summary for Form1
      ///
      /// WARNING: If you change the name of this class, you will need to change the
      ///          'Resource File Name' property for the managed resource compiler tool
      ///          associated with all .resx files this class depends on.  Otherwise,
      ///          the designers will not be able to interact properly with localized
      ///          resources associated with this form.
      /// </summary>
      public ref class Form1 : public System::Windows::Forms::Form
      {
      public:
       CCyUSBDevice *USBDevice, *CyStreamdev;
        int AltInterface;
        bool bPnP_Arrival;
        bool bPnP_Removal;
       bool bPnP_DevNodeChange;
      private: System::Windows::Forms::Button^  button1;
      public:
      public:
      private: System::Windows::Forms::ListBox^  listBox1;
      public:

            Form1(void)
            {
                  InitializeComponent();
                  USBDevice =new CCyUSBDevice((HANDLE)this->Handle,CYUSBDRV_GUID,true);
                  RefreshList();
                  //
                  //TODO: Add the constructor code here
                  //
            }
       virtual void WndProc( Message% m ) override
       {
         if (m.Msg == WM_DEVICECHANGE)
         {
             // Tracks DBT_DEVNODES_CHANGED followed by DBT_DEVICEREMOVECOMPLETE
            if (m.WParam == (IntPtr)DBT_DEVNODES_CHANGED)
            {
                bPnP_DevNodeChange = true;
                bPnP_Removal = false;
            }
             // Tracks DBT_DEVICEARRIVAL followed by DBT_DEVNODES_CHANGED
            if (m.WParam == (IntPtr)DBT_DEVICEARRIVAL)
            {
                bPnP_Arrival = true;
                bPnP_DevNodeChange = false;
            }
            if (m.WParam == (IntPtr)DBT_DEVICEREMOVECOMPLETE)
                bPnP_Removal = true;
```

OBSOLETE

```
        // If DBT_DEVICEARRIVAL followed by DBT_DEVNODES_CHANGED
        if (bPnP_DevNodeChange && bPnP_Removal)
        {
            bPnP_Removal = false;
            bPnP_DevNodeChange = false;
            GetDevice();
         }
        // If DBT_DEVICEARRIVAL followed by DBT_DEVNODES_CHANGED
         if (bPnP_DevNodeChange && bPnP_Arrival)
         {
             bPnP_Arrival = false;
             bPnP_DevNodeChange = false;
             GetDevice();
          }
        }
        Form::WndProc( m );
    }
       void GetDevice()
       {
           USBDevice = new CCyUSBDevice((HANDLE)this->Handle,CYUSBDRV_GUID,true);
           AltInterface = 0;
           if (USBDevice->DeviceCount())
           {
              Text = "Device Attached";
           }
           else
           {
              Text = "Device Not Attached";
           }
           RefreshList();
       }
protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~Form1()
        {
               if (components)
               {
                      delete components;
               }
        }

private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container ^components;
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {
        CyStreamdev -> SetAltIntfc (++AltInterface);
        Text = Convert::ToString (CyStreamdev->AltIntfc());

}
 void RefreshList()
{
        int vid, pid;
        CyStreamdev = NULL;
        button1->Enabled = FALSE;

        listBox1->Items->Clear();
        listBox1->Text = " ";
        for(int i=0;i<USBDevice->DeviceCount();i++)
        {
           USBDevice->Open(i);
```

OBSOLETE

```
                listBox1->Items->Add(gcnew String(USBDevice->FriendlyName));
                listBox1->Text= Convert::ToString(listBox1->Items[i]);

                vid = USBDevice -> VendorID;
                pid = USBDevice -> ProductID;
                    if(vid == 0x04B4 && pid == 0x1003)
                    {
                          CyStreamdev = new CCyUSBDevice((HANDLE)this->Handle,
                          CYUSBDRV_GUID,true);
                          button1 ->Enabled = TRUE;
                    }
            }
        }

    #pragma region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            void InitializeComponent(void)
            {
                this->button1 = (gcnew System::Windows::Forms::Button());
                this->listBox1 = (gcnew System::Windows::Forms::ListBox());
                this->SuspendLayout();
                //
                // button1
                //
                this->button1->Location = System::Drawing::Point(505, 19);
                this->button1->Name = L"button1";
                this->button1->Size = System::Drawing::Size(75, 23);
                this->button1->TabIndex = 0;
                this->button1->Text = L"button1";
                this->button1->UseVisualStyleBackColor = false;
                this->button1->Click += gcnew System::EventHandler(this,
                &Form1::button1_Click);
                //
                // listBox1
                //
                this->listBox1->FormattingEnabled = true;
                this->listBox1->Location = System::Drawing::Point(19, 19);
                this->listBox1->Name = L"listBox1";
                this->listBox1->Size = System::Drawing::Size(457, 186);
                this->listBox1->TabIndex = 2;
                this->listBox1->SelectedIndexChanged += gcnew System::EventHandler(this,
                &Form1::listBox1_SelectedIndexChanged);
                //
                // Form1
                //
                this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                this->ClientSize = System::Drawing::Size(620, 266);
                this->Controls->Add(this->listBox1);
                this->Controls->Add(this->button2);
                this->Controls->Add(this->button1);
                this->Name = L"Form1";
                this->Text = L"Form1";
                this->ResumeLayout(false);

            }
    #pragma endregion

};
}
```

OBSOLETE

Document History

Document Title: Introduction to CyAPI.lib Based Application Development Using VC++ - AN61744

Document Number: 001-61744

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2934442	CPPK	05/20/10	New Application Note
*A	3007276	CPPK	08/13/10	Title is changed from 'Developing USB Applications with VC++' to 'Getting Started With USB Application Using VC++'.
*B	3088800	CPPK	11/17/10	Changed title to "Getting started with SuiteUSB Applications using VC++".
*C	3186856	CPPK	03/03/11	Updated the title and the abstract.
*D	3600853	GAYA	04/30/2012	Fixed links to pictures and other documentation, added pointers to advanced examples. Added Appendix with entire code for reference. Attached project files for VS 2008/VS 2010.
*E	3974573	NIKL	04/19/2013	Obsolete document.

OBSOLETE

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

OBSOLETE

All trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2010-2013. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.