# Developing USB Applications with VC++

# AN61744

**Author**: Praveen Kumar C P
**Associated Project**: Yes
**Associated Part Family**: CY7C68013/ CY7C68013A
**Software Version**: Microsoft Visual Studio 2008
**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, which contains *CyAPI.lib*. *CyAPI.lib* provides a powerful C++ programming interface to USB devices. More specifically, it is a C++ class library that provides a high-level programming interface to the *CyUsb.sys* device driver. The library is able to communicate only with USB devices that are served by this driver. The focus of this article includes a history and walk-through of writing your first USB application. The USB application explained here is a Windows Forms application. *CyAPI.lib* can also be used to write an MFC application (unmanaged code), that is not covered in this article.

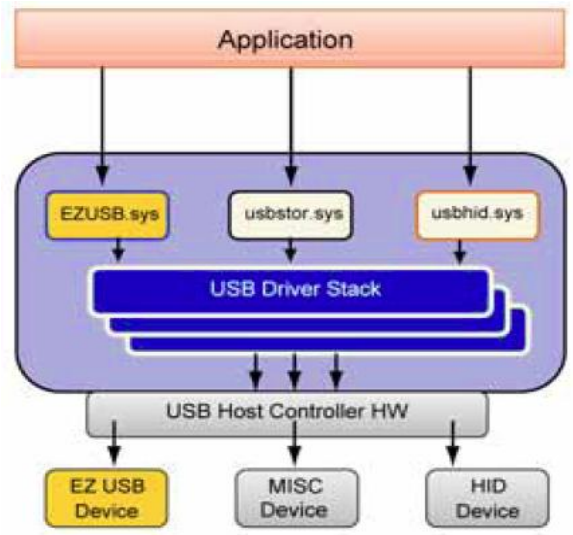
## Introduction

Applications' communication with USB devices have evolved dramatically. In the early days, the application writing process involved making direct calls to drivers. 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 accessing USB devices very difficult.

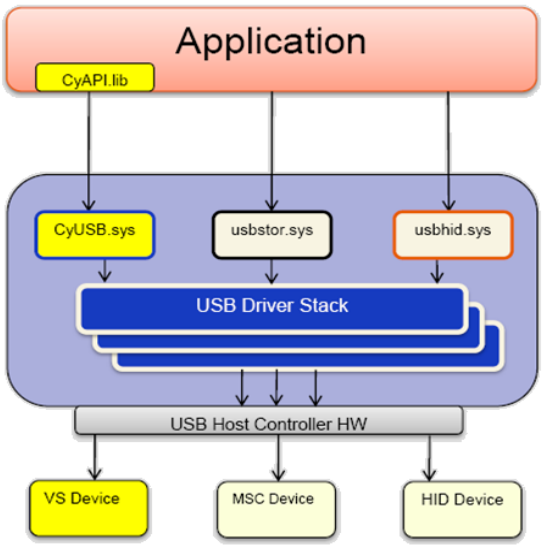
Cypress released *CyAPI.lib*, first in its USB Developers' µStudio and then in the latest Cypress SuiteUSB which provided a high level programming interface to get a device handle and communicate with Cypress USB devices.

*CyAPI.lib* automatically takes care of activities such as error handling which otherwise had to be taken care by the user when making direct calls to the drivers. *CyAPI.lib* was 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.

**Early Days**




**New CyAPI.lib**



*CyAPI.lib* is a statically linked C++ library, so its classes and functions can be accessed from C++ compilers like 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.

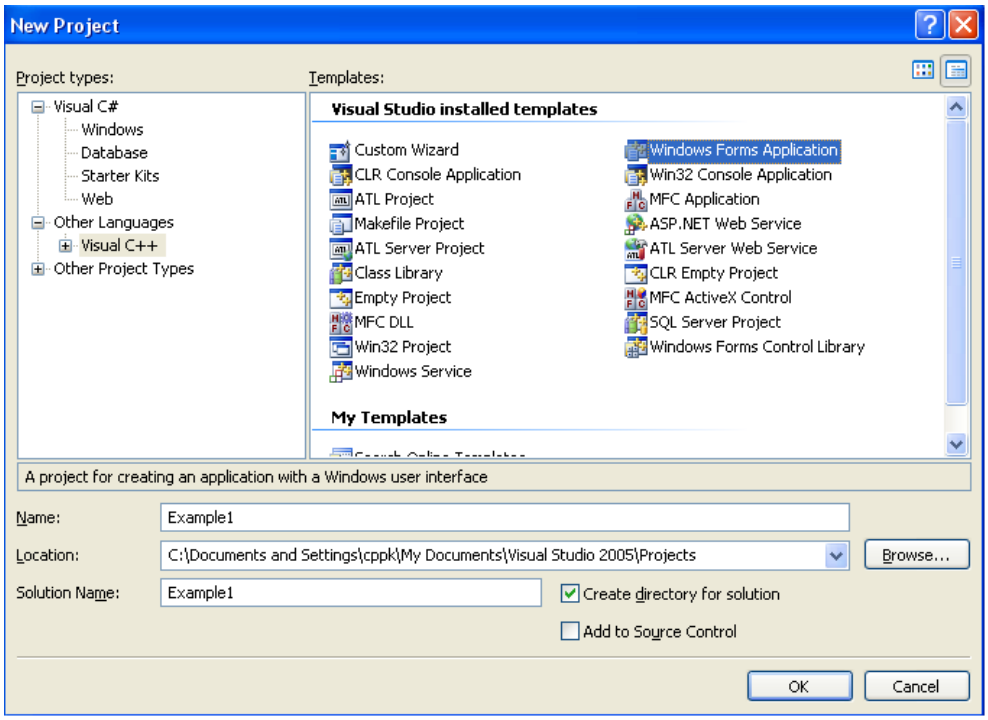
The following topic walks you through developing your first application with *CyAPI.lib*. The following examples are written in Visual C++.

## Writing Your First Application

Before you begin writing your application, ensure that you have installed *SuiteUSB.Net*, then start a new project in Visual Studio 2008 by clicking on **File** > **New** > **Project**. 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 1.

Click **OK** and a blank form displays. This form is a functional application; click on the green arrow to start the application.

Figure 1. Starting a New Project.




1. Right click on Source Files and select **Add** > **Existing Item**, under the Solution Explorer window. Browse to the installation directory of SuiteUSB 3.4.1 and double click *CyAPI.lib.* This references the library to your project.
2. If blank form displays, right click outside in the white space and click **View Code**. This is your code view window and displays the initial code that Microsoft puts in to start your project.
3. At the top, you can 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.
4. After adding the reference to *CyUSB.lib*, you have to expose the interface to it. This can be done by including a line CyAPI.h which gives you access to the library's APIs, classes, and other functionality. This is done in the following two 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:\Program Files\Cypress
   Semiconductor\Cypress Suite
   USB\CyAPI\inc
   ```

   after the installation of SuiteUSB. This folder contains the *CyAPI.h* file. Click **OK.**

   b. Add a line

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

   after the lines added in step 4.

5. Go to **Project** > **Properties**. In the dialog box, select **Configuration Properties** > **Linker** > **Input** > **Additional Dependencies** and type **user32.lib**.
6. 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)**.
7. Insert the following code into your application at the exact location in the Form1 class.

```
1. public ref class Form1 : public System::Windows::Forms::Form
2. {
3. public:
4. CCyUSBDevice *USBDevice;
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 = "Device Not Attached";
60.    }
61. }
```

## Application Code Analysis

The CCyUSBDevice class is at the heart of the CyUSB class library. To utilize the library, a working knowledge of the CCyUSBDevice class is essential. The CCyUSBDevice class is the primary entry point into 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).

We 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 have all been properly initialized and are ready for 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.

Whenever a device has been 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 has been 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 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 makes use of the DeviceCount() function (line number 53) which is a member of CCyUSBDevice class. DeviceCount() function returns the number of devices attached to the *CyUSB.sys* driver. If this function returns a non-zero value, we can conclude that there are one or more devices connected to the host which are 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 = "Device Not Attached".
   ```
   These lines of code display the status correctly only when we have a single device connected (or removed) to the host. 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 above mentioned code, the word **Form1** is displayed which is not very informative. 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 wrote 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 some advanced features that are used to make the application more productive.

## Advanced Features

Before proceeding to advanced features, a more detailed discussion about *CyAPI.lib* is required. CCyUSBDevice provides two main components (an indepth 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 you need in your application, including functions such as GetDeviceDescriptor(), Reset() and SetAltIntfc(); properties such as DeviceName, DevClass, VendorID, ProductID. The first application you wrote allowed you to detect PnP 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 that the Windows Device Manager detects it as a CyStream device.

## Add Buttons and Toggle a 7-Segment Display

1. Click **Form1.h [Design]** tab in Visual Studio.
2. Click **View** > **Toolbox**.
3. In the Toolbox, click and drag the **Button** anywhere on your application.
4. A button labeled **Button1** appears on your form. Double click the **Button**.
5. An event handler is created. Anytime a user clicks 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 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:

   ```
   USBDevice->SetAltIntfc (AltInterface);
   Text = Convert::ToString
   (AltInterface++);
   ```

   As discussed before, SetAltIntfc (UCHAR alt) is a function that is used to set the alternate interface setting for the device to the value alt. Since the *CyStream.iic* firmware displays the value of the alternate interface on 7-segment display of the FX2LP DVK, 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, where a USB device has multiple alternate interfaces each operating with its own set of endpoints and the current interface being used can be displayed on the screen.
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 **listBox** onto the form and expand it to take up most of the room on the form.
3. Right click in the white space outside of your form and click **View Code**.
4. Insert the following code to Form1.

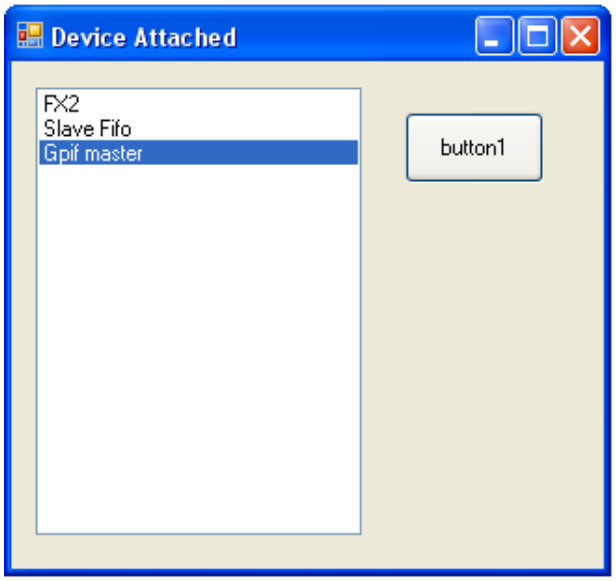
   ```
   1. void RefreshList()
   2. {
   3.     listBox1->Items->Clear();
   4.     listBox1->Text = " ";
   5. for(int i=0;i<USBDevice-
      >DeviceCount();i++)
   6.     {
   7.         USBDevice->Open(i);
   8.         listBox1->Items->Add(gcnew
      String(USBDevice->FriendlyName));
   9.      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

   This Function gets all the devices connected to CyUSB.sys driver and displays their names in a listbox. DeviceCount() is a function implemented in *CyAPI.lib* which 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 so as to keep the list of devices updated. 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.
5. Add a single line of code

   ```
   RefreshList();
   ```

   to call the above function in GetDevice(). Your view should look similar to the following.




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

## About the Author


| | |
|---|---|
| **Name:** | Praveen Kumar C P |
| **Title:** | Application Engineer |
| **Contact:** | cppk@cypress.com |

# Document History

**Document Title: Developing USB Applications with VC++**

**Document Number: 001-61744**

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | 2934442 | CPPK | 05/20/2010 | New Application Note |

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