USB HID (human interface device) class — это устройства с USB интерфейсом, разработанные для взаимодействие межу человеком и компьютером. К таким устройствам относятся клавиатуры, мышки, игровые джойстики, так же к таким устройствам можно отнести устройства на микроконтроллерах.

Большим преимуществом данного класса USB — это наличие огромного количество драйверов, что дает возможность повсеместно использовать именно этот класс, не привязываясь к конкретному устройству.

В языке **C#** есть несколько библиотек для работы с USB HID. На мой взгляд самой простой и удобной библиотекой является **UsbLibrary.dll.**

Для начала работы с данной библиотекой в приложении C# необходимо добавить на нее ссылку в проект. Для этого в Обозревателе решений нажимаем правой клавишей мышки на поле `Ссылки` и в всплывающем меню нажимаем `Добавить ссылку`. Далее в появившемся окне нажимаем кнопку `Обзор` и выбираем файл `UsbLibrary.dll.` Нажимаем кнопку `Ок` и все, библиотека добавлена, теперь ее можно использовать в проекте C#.

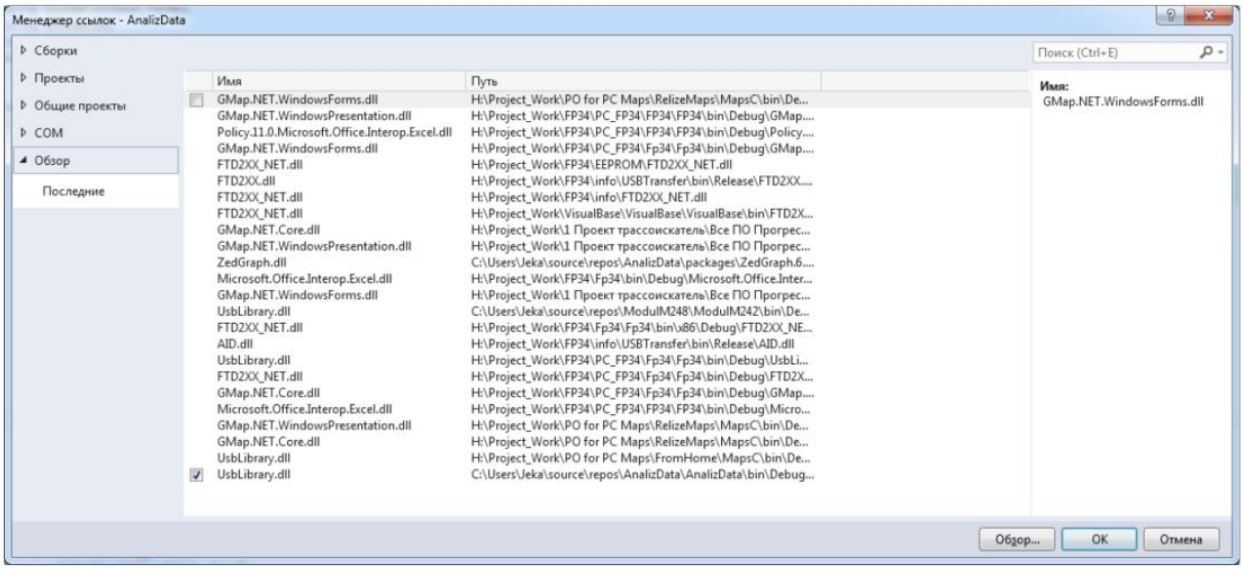


Рисунок 1. Добавление ссылки на библиотеку

Файл DLL библиотеки размещайте в каталоге программы, чтобы проще было ее переносить.

Для начала необходимо добавить библиотеку директивой `using`:

```
using UsbLibrary;
```

Далее объявляем переменную класса **UsbHidPort**.

```
UsbHidPort usb = new UsbHidPort();
```

Затем объявите переменные **USBDevVendorID**, **USBDevProductID** и присвойте им необходимые значения. Для моего устройства это **0x0045** и **0x1307**. Так как я сам писал хост для устройства USB HID, то **VID** и **PID** указывал сам.

```
private static byte[] USBSend = new byte[65];
private UInt16 USBDevProductID = 0x0045;
private UInt16 USBDevVendorID = 0x1307;
```

Если не известно какой VID и PID устройства то идем в диспетчер в раздел **Устройства HID**

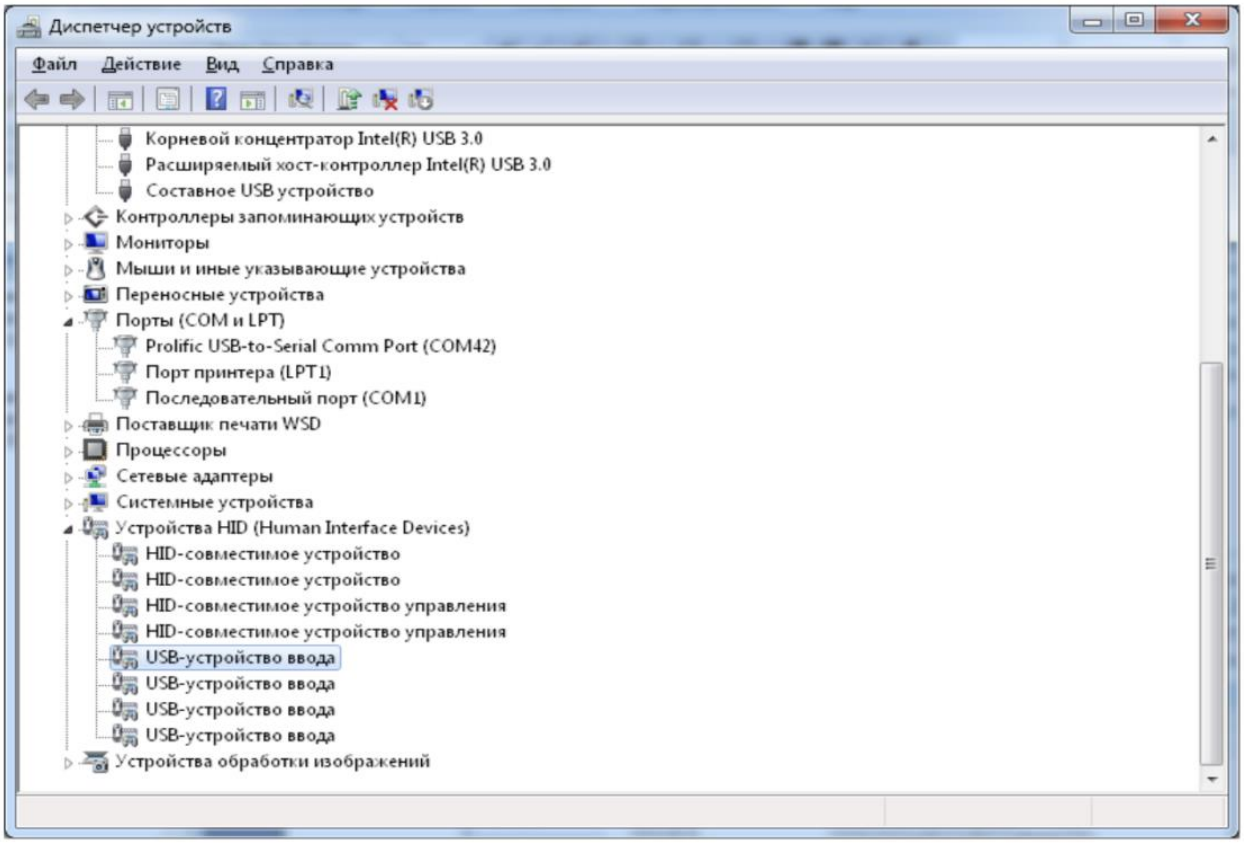


Рисунок 2. Устройства HID в Диспетчере устройств

Выбираем наше устройство и нажимаем правой клавишей мыши и в новом окне в пункте **Сведения,** необходимо выбрать **ИД оборудования.** Отображенные значения и будут наши PID и VID;

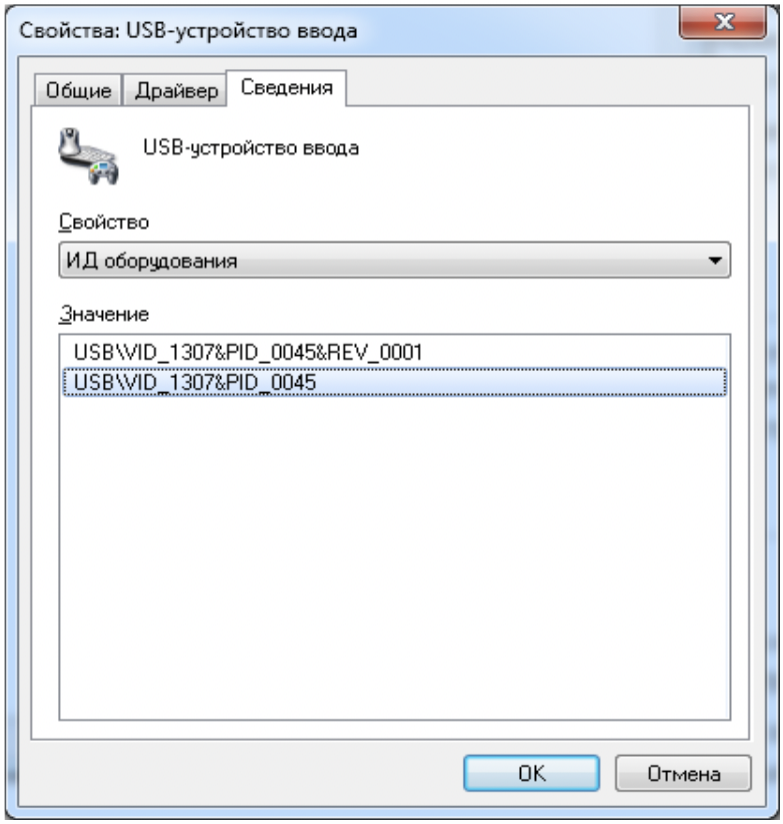


Рисунок 3. Свойства USB-устройства ввода

Ниже представлен код использования USB HID.

```
private void Form1_Load(object sender, EventArgs e)
{
  // переопределяем события класса USB HID
  // Событие на поключение устройства
  usb.OnSpecifiedDeviceArrived += usb_OnSpecifiedDeviceArrived;
  // событие когда устройство отключается
  usb.OnSpecifiedDeviceRemoved += usb_OnSpecifiedDeviceRemoved;
  //событие на прием данных
  usb.OnDataRecieved += usb_OnDataRecieved;
  usb.ProductId = USBDevProductID;
  usb.VendorId = USBDevVendorID;
}
private void usb_OnSpecifiedDeviceArrived(object sender, EventArgs e)
```

```
{
  Connecting = true;
}
private void usb_OnSpecifiedDeviceRemoved(object sender, EventArgs e)
{
  Connecting = false;
}
```

Ниже представлен пример из реального работающего кода, по приему данных с устройства USB Hid. Суть такова: из программы на C# пользователь по таймеру отправляет команду на HID устройство, функция `OnDataRecieved` принимает данные, отправленные устройством. Если в данных не встречается строка **«MESSAGE END»** добавляем пакет данных в `List data`.

```
private void timWork_Tick(object sender, EventArgs e)
{
  int cell = 0;  // целое число
  int doubl = 0; // дробное число
  byte[] USB_Read = new byte[65];
  if (usb.Ready())
  {
    USBSend[1] = Convert.ToByte('p');                    // обработать
исключение
    this.usb.SpecifiedDevice.SendData(USBSend);
    while (EndMessege == 0) Application.DoEvents();
    if (EndMessege == 1)
    {
      // Если есть данные от прибора обрабатываем их и выводим график
      if (data.Count != 0)
      {
        USB_Read = data[0];
        cell = 0;  // целое число
        doubl = 0; // дробное число
```

```
        cell = Combine(USB_Read[12], USB_Read[13]);
        list_SensorE.Add(list_SensorE.Count + 1, Convert.ToDouble(cell));
        // Если еденица то записываем в файл
        if (flagDateWrite == 1)
        {
          String DataWrite;
          DataWrite = list_SensorE[list_SensorE.Count - 1].ToString()
          using (StreamWriter fstream = new StreamWriter("test.txt", true,
Encoding.GetEncoding(1251)))
          {
            // операции с fstream
            fstream.WriteLine(DataWrite);
          }
        }
      }
      DrawGraf();
      Application.DoEvents(); // Чтобы программа не зависала;
      data.Clear();
      EndMessege = 0;
    }
  }
  Application.DoEvents(); // Чтобы программа не зависала;
}
private void usb_OnDataRecieved(object sender,
UsbLibrary.DataRecievedEventArgs args)
{
  string tempstr = "";
  byte[] tempAr = new byte[64];
  byte[] UsbR = new byte[65];
  UsbR = args.data;
```

```
  j++;
  if (EndMessege == 0)
  {
    for (int i = 0; i <= tempAr.Length - 1; i++)
    {
      tempAr[i] = UsbR[i + 1];
    }
    temp = System.Text.Encoding.ASCII.GetString(tempAr);
    for (int i = 0; i <= 10; i++)
    {
      tempstr = tempstr + temp[i];
    }
    if (tempstr == "MESSAGE END")
      { EndMessege = 1; }
    else { data.Add(tempAr); }
  }
}
```

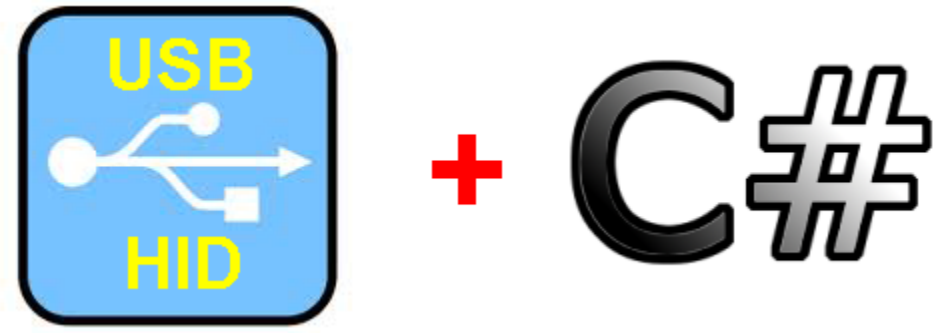
Скачать библиотеку UsbLibrary для C#

Записи по теме

Библиотека HID USB Library для Visual Studio .Net (C#, Visual Basic)

Добавил(а) microsin

Перевод статьи, посвященной удобной библиотеке **HID USB Driver** / **Library** for **.Net** / **C#** ("HID USB Driver / Library for .Net / C#", автор Florian Leitner-Fischer).



[**Почему появилась библиотека**]

Нужно было найти новое решение чтобы подключить нашу аппаратуру к **PC** (компьютеру) через **USB**, потому что результаты, которые мы получили с помощью чипов **FTDI**, нас полностью не устраивали.

Выбранный нами контроллер USB Maxim **MAX3420E** был подключен к микроконтроллеру через Serial Peripheral Interface (**SPI**). Поставляется MAX3420E без каких-либо драйверов или библиотек, через которые Вы могли бы общаться с ним. По этой причине и появилась библиотека, описанная в статье.

После целого дня, потраченного на гугление и чтение полученной информации я решил, что самый простой и лучший метод - использовать **Human Interface Device Class** (**HID**). HID-Class является стандартным классом устройства (generic device class), так что все устройства этого класса поддерживаются Microsoft Windows из коробки, и имеются готовые **DLL** с функциями, которые можно использовать для доступа к устройствам USB HID.

Стартовой точкой для "драйвера| была статья, которую я нашел на Microsoft developer network (**MSDN**): "Is That You? Writing Better Software for Cool USB Hardware" (Это для Вас? Написание лучшего программного обеспечения для хорошей аппаратуры USB), написанную Scott Hanselman. Программное обеспечение автора основывалось на **USBSharp Class** [1].

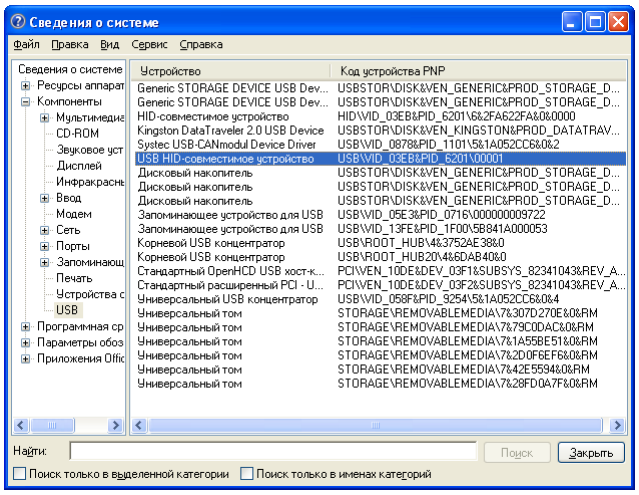
К счастью, с MAX3420E был пример кода, который конфигурирует контроллер как Human Interface Device, так что мы не испытывали особых затруднений, чтобы понять, как сконфигурировать его в режиме HID.

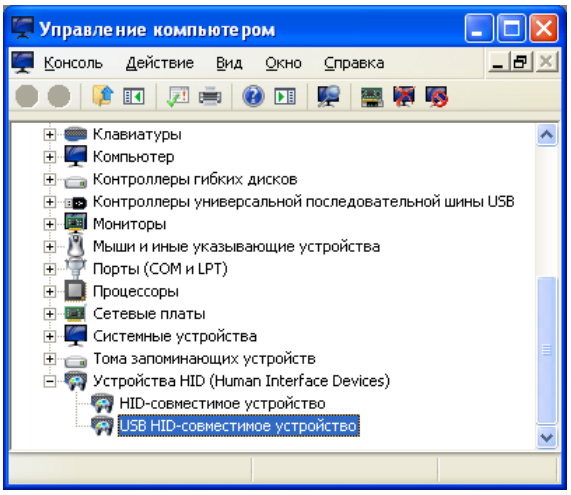
[**Human Interface Device Class, HID**]

Как упоминалось выше, HID Class является стандартным классом устройства, драйвер для которого интегрирован в операционную систему, что упрощает работу с ним. Если, к примеру, в операционную систему Windows подключено новое устройство HID, то ему не требуется никакая инсталляция драйвера. Функции для получения доступа и управления устройством HID включены в Windows **hid.dll**, которая расположена в папке System32.

Если не знаете наверняка, что Ваше устройство - устройство HID, то взгляните на небольшое приложение "Сведения о системе". Оно является частью windows, и Вы можете запустить его командой start -> run "**msinfo32.exe**". Эта программа покажет в разделе Компоненты -> USB информацию о подключенных устройствах USB. Для обновления списка устройств при переподключениях выбирайте Вид -> Обновить. Кроме того, сведение о подключенных устройствах

USB HID можно получить в Диспетчере Устройств, см. Панель Управления -> Администрирование -> Управление компьютером -> Устройства HID (Human Interface Devices).





[**Специфика USB: идентификация устройства USB**]

Устройство USB идентифицируется по его ID вендора (**Vendor ID**, **VID**) и ID продукта (**Product ID**, **PID**). Именно по этим идентификаторам ПО хоста подключается к конкретному устройству USB (из этого следует, кстати, что нельзя одновременно подключиться к двум устройствам USB HID, если они имеют одинаковые VID и PID). Эти ID состоят из префикса ("vid_" для vendor id или "pid_" для product id) и 4 шестнадцатеричных цифр. Например, MAX3420E имеет vendor id vid_06ba и product id pid_5346. Обычно оба значения могут быть изменены в исходном коде USB-устройства (подразумевается, что Вы имеете доступ к этому коду).

[**Размер пакета / скорость передачи данных на соединении**]

Устройства HID обмениваются с PC данными через репорты HID (**HID report**). Эти репорты могут содержать от 8 до 64 байт данных. Каждую миллисекунду один репорт отправляется от PC к устройству USB и обратно. Это означает, что теоретическая максимальная скорость обмена может достигать 64000 байт/сек.

[**Интерфейс библиотеки HID USB**]

Библиотека написана на **C#** и разработана в виде **DLL**, что позволяет её очень просто интегрировать в новый проект. Библиотека рассчитана на устройства USB HID с размером 64 байта для конечных точек ввода и вывода (**endpoints**). Просто импортируйте DLL в проект, и получите удобный доступ к устройствам USB HID. Я старался сделать программный интерфейс библиотеки максимально простым, при сохранении максимально возможного функционала. В настоящий момент имеются следующие функции:

USBInterface(String vid, String pid) - метод конструктора, инициализирующий новый экземпляр (переменную) класса. У конструктора в этом случае два параметра - идентификатор вендора и идентификатор продукта USB-устройства. По этим параметрам будет происходить поиск нужного USB HID устройства.

USBInterface(String vid) - метод конструктора, инициализирующий новый экземпляр (переменную) класса. У конструктора в этом случае один параметр - идентификатор вендор. По этому параметру будет происходить поиск нужного USB HID устройства.

Connect() - этот метод устанавливает соединение до устройства USB. Вы можете быть уверенным, что подключились к нужному устройству только в том случае, если использовали конструктор USBInterface с двумя параметрами - vid и pid. Если же Вы использовали конструктор с одним параметром vid, то может произойти соединение с другим устройством с тем же самым Vendor ID (VID) - не с тем, с что ожидали, и Вы не сможете определить, к какому устройству произойдет подключение. Метод Connect возвратит false, если произошла любая ошибка.

Disconnect() - метод производит отключение от устройства.

getDeviceList() - метод возвращает список устройств с Vendor ID (VID), указанным в конструкторе. Эта функция понадобится Вам чтобы узнать, сколько подключено USB устройств (и какие они) с указанным идентификатором вендора. Возвращает список строк (String list) с системными путями до устройств.

write(Byte[]) - метод записывает массив байт в устройство USB. Если размер массива превышает 64, то массив будет поделен на несколько массивов, каждый по 64 байта. Массивы будут отправлены друг за другом, байты 0..63 будут отправлены первыми, затем будут отправлены байты 64..127, и так далее. Метод возвратит true, если все байты будут успешно записаны.

startRead() - метод используется для активизации "состояния чтения" (reading-State). Если Вы выполните эту команду, то запустится поток, который прослушивает устройство USB и ждет появления данных.

stopRead() - метод переключает библиотеку из состояния "reading-State" в состояние ожидания (idle-State). Путем выполнения этой команды поток чтения данных останавливается, и теперь данные могут быть получены.

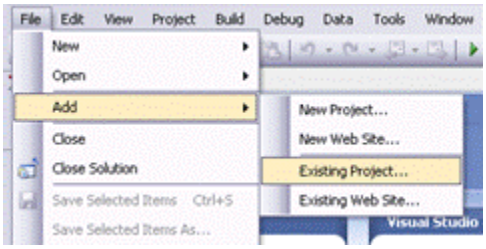
enableUsbBufferEvent(EventHandler) - путем вызова этого метода с обработчиком события (System.EventHandler) добавляется событие прослушивания буфера USB. Таким образом будет вызван обработчик события всякий раз, когда набор данных добавлен к буферу (и также получен от USB-устройства).

[**Как интегрировать библиотеку HID USB в проект Visual Basic**]

Есть два способа интеграции HID USB Library в проект Visual Studio. Один из них - добавление проекта библиотеки к Вашему решению Visual Studio solution. Второй способ - добавление ссылки на **USBHIDDRIVER.dll** в проект, где Вы хотите использовать библиотеку HID USB. Предварительно скачайте пакет библиотеки [2], и распакуйте его файлы в Ваш проект приложения.

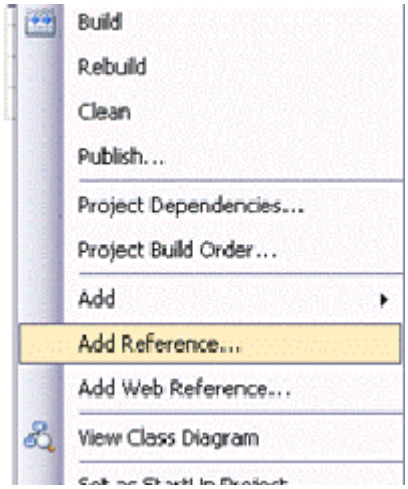
Добавление проекта библиотеки в решение (solution) Visual Studio 2005, процесс по шагам:

1. Откройте Ваше решение (Visual Studio solution). Выберите в меню File -> Add -> Existing Project.

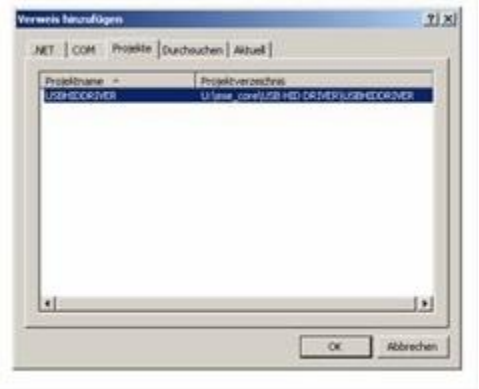


2. Откроется диалог открытия файла. Найдите файл USBHIDDRIVER.csproj и откройте его (кликните Open).

3. Правым щелчком мыши на проекте Visual Studio откройте контекстное меню проекта и выберите Add Reference (добавить ссылку).

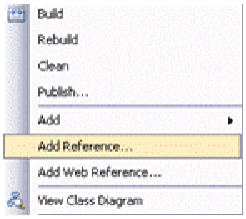


4. Выберите закладку Projects, затем выберите USBHIDDRIVER и щелкните OK.



Добавление USBHIDDRIVER.dll к проекту Visual Studio, процесс по шагам.

1. Правым щелчком мыши на проекте Visual Studio откройте контекстное меню проекта и выберите Add Reference (добавить ссылку).



2. Выберите закладку Browse, найдите USBHIDDRIVER.dll (или USBHIDDRIVER.dll.dll) и кликните OK.

[**Использование USBHIDDRIVER.dll вместе с Visual Basic 6**]

Исполняемый код .Net (.Net runtime) позволяет необслуживаемым клиентам COM (**unmanaged COM aware clients**, наподобие приложений Visual Basic 6) получить доступ к компонентам .Net через взаимодействие COM (COM interop) и инструменты, предоставленные средой разработки.

Библиотека USBHIDDRIVER.dll является сборкой .NET, к которой не может быть получен доступ из Visual Basic 6. Поэтому мы должны создать библиотеку типов, которая может использоваться с Visual Basic 6. Это можно сделать с помощью инструмента **RegAsm.exe**, который Вы можете найти в директории инсталляции среды разработки .Net. Создайте командный файл **BAT** в папке, где находится USBHIDDRIVER.dll, впишите в него следующую команду и затем запустите его:

```
"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe" USBHIDDRIVER.dll.dll /tlb:
USBHIDDRIVER.dll.tlb
```

Теперь Вы должны скопировать и **dll**, и **tlb** файлы в том же каталоге в папку приложения, Visual Basic, чтобы оно могло использовать библиотеку USB HID.

[**Как использовать HID USB Library**]

Предположим, Вам нужно наладить обмен с USB-устройством, которое имеет vendor id vid_06ba и несколько возможных product id. Первое, что Вам понадобится - список всех подключенных к PC устройств, которые имеют указанный vendor id. Чтобы получить этот список - просто создайте новый экземпляр класса USBInterface, при этом укажите конструктору только один параметр - VID (vendor id). Затем вызовите метод getDeviceList, и получите список подключенных устройств.

```
USBHIDDRIVER.USBInterface usb = new USBInterface("vid_06ba");
String[] list = usbI.getDeviceList();
```

Для организации обмена с нужным устройством из списка сделайте новый экземпляр USBInterface, и на этот раз укажите конструктору два параметра - VID и PID. Затем вызовите метод Connect, и получите соединение с USB-устройством.

```
USBHIDDRIVER.USBInterface usb = new USBInterface("vid_06ba", "pid_5346");
usb.Connect();
```

Как только соединение с устройством установлено, Вы можете читать или записывать данные. Для записи данных просто вызовите метод write с указанием в параметре массива байт, который нужно передать. Чтение может быть реализовано двумя способами: первый без события USB buffer event, и второй с событием USB buffer event.

Если разрешено событие, то обработчик события (event handler method) будет вызван всякий раз, когда будет принят новый набор данных. В другом случае Вы можете обработать данные по таймеру наподобие примерно следующего кода:

```
usb.write(startByte);
usb.enableUsbBufferEvent(new System.EventHandler(myEventCacher));
Thread.Sleep(5);
usb.startRead();
…
usb.stopRead();
usb.write(stopByte);
```

[**Что происходит внутри HID USB Library**]

Библиотека основана на классе **USBSharp**, который импортирует все нужные методы из **kernel32.dll** и **hid.dll**. Класс **HIDUSBDevice** оборачивает эти методы и организует чтение в отдельном потоке. Главным интерфейсом библиотеки является класс **USBInterface**, который доступен снаружи dll. В классе USBInterface существует объект **ListWithEvent**, который основывается на **ArrayList** с бонусом в виде запуска обработчика события, когда добавляется новый набор данных. Наконец, что не менее важно, в классе **HidApiDeclaration** Вы найдете некоторые объявления, необходимые для работы с USB HID.

[**Примечание переводчика**]

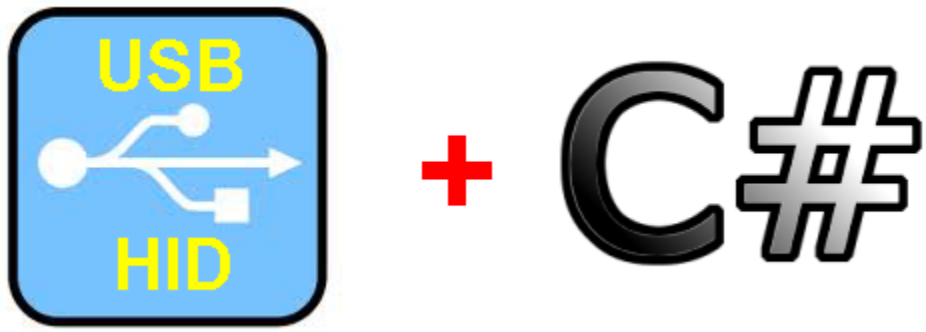
В Интернете можно найти очень похожий аналог библиотеки, описанной в этой статье, только не в виде DLL, а в виде файлов исходного кода C# (модулей .cs) [3]. Библиотека имеет класс **UsbLibrary.UsbHidPort**, в котором сосредоточен весь интерфейс работы с устройством USB HID. Возможностей у этой библиотеки больше - например, есть очень удобные события подключения и отключения устройства USB, благодаря чему легко реализовать прозрачную работу с устройством USB HID и восстанавливать обмен данными с USB-устройством без необходимости перезапуска программы.

[**Ссылки**]

1. The HID Page - страничка, посвященная классу HID.
2. Пакет библиотеки HID USB Driver / Library for .Net / C# (USBHIDDRIVER.dll).
3. UsbLibrary.UsbHidPort class source code.
4. Библиотеки для управления устройствами USB HID.
5. USB in a NutShell - путеводитель по стандарту USB.
6. HID API для Linux, Mac OS X и Windows.

Thêm bởi microsin

Bản dịch bài viết về thư viện tiện dụng **HID USB Driver** / **Library** for **.Net** / **C#** ("HID USB Driver/Library for .Net/C#" của Florian Leitner-Fischer).



[**Tại sao thư viện xuất hiện**]

Cần phải tìm một giải pháp mới để kết nối thiết bị của chúng tôi với **PC** (máy tính) qua **USB** , bởi vì kết quả mà chúng tôi thu được khi sử dụng chip **FTDI** không hoàn toàn khiến chúng tôi hài lòng.

Bộ điều khiển USB Maxim **MAX3420E** mà chúng tôi chọn đã được kết nối với bộ vi điều khiển thông qua Giao diện ngoại vi nối tiếp (**SPI**). MAX3420E xuất xưởng mà không có bất kỳ trình điều khiển hoặc thư viện nào để bạn có thể giao tiếp với nó. Vì lý do này, thư viện được mô tả trong bài viết đã xuất hiện.

Sau cả ngày tìm kiếm và đọc thông tin nhận được, tôi quyết định rằng phương pháp dễ nhất và tốt nhất là sử dụng **Lớp thiết bị giao diện con người** (**HID**). HID-Class là một lớp thiết bị chung, vì vậy tất cả các thiết bị trong lớp này đều được Microsoft Windows hỗ trợ ngay từ đầu và có các tệp **DLL** được tạo sẵn với các chức năng có thể được sử dụng để truy cập các thiết bị USB HID.

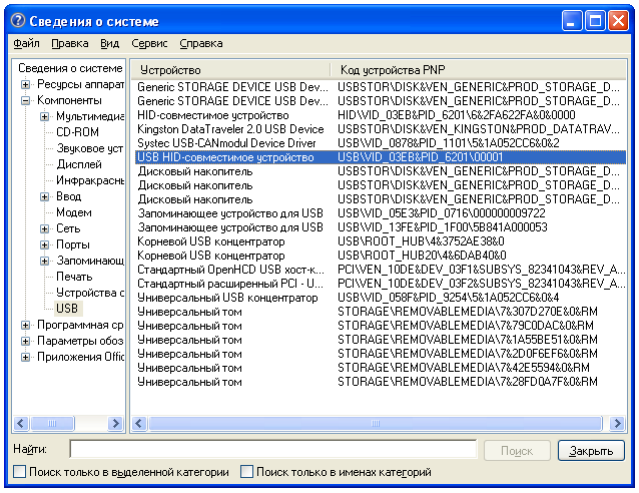
Điểm khởi đầu cho "driver|" là một bài viết tôi tìm thấy trên mạng lưới nhà phát triển Microsoft (**MSDN**): "Is That You? Viết phần mềm tốt hơn cho phần cứng USB tuyệt vời" của Scott Hanselman. Phần mềm của tác giả dựa trên **USBSharp Class** [1].

May mắn thay, MAX3420E đi kèm với mã mẫu định cấu hình bộ điều khiển làm Thiết bị giao diện con người, vì vậy chúng tôi không gặp quá nhiều khó khăn khi tìm ra cách định cấu hình bộ điều khiển ở chế độ HID.

[**Lớp thiết bị giao diện con người, HID**]

Như đã đề cập ở trên, HID Class là loại thiết bị tiêu chuẩn mà trình điều khiển được tích hợp vào hệ điều hành, giúp làm việc dễ dàng hơn. Ví dụ: nếu một thiết bị HID mới được kết nối với hệ điều hành Windows thì thiết bị đó không yêu cầu cài đặt bất kỳ trình điều khiển nào. Chức năng truy cập và quản lý thiết bị HID được bao gồm trong Windows **hid.dll** , nằm trong thư mục System32.

Nếu bạn không biết chắc chắn rằng thiết bị của mình có phải là thiết bị HID hay không, hãy xem ứng dụng Thông tin hệ thống nhỏ. Nó là một phần của windows và bạn có thể chạy nó bằng lệnh start -> run " **msinfo32.exe** ". Chương trình này sẽ hiển thị trong Thành phần -> Thông tin USB về các thiết bị USB được kết nối. Để cập nhật danh sách thiết bị khi kết nối lại, chọn Xem -> Làm mới. Ngoài ra, có thể lấy thông tin về các thiết bị USB HID được kết nối trong Trình quản lý thiết bị, xem Bảng điều khiển -> Công cụ quản trị -> Quản lý máy tính -> Thiết bị HID (Thiết bị giao diện con người).



[**USB cụ thể: Nhận dạng thiết bị USB**]

Thiết bị USB được xác định bởi ID nhà cung cấp (**Vendor ID** , **VID**) và ID sản phẩm (**Product ID** , **PID**). Nhờ những mã nhận dạng này mà phần mềm máy chủ kết nối với một thiết bị USB cụ thể (nhân tiện, bạn không thể kết nối với hai thiết bị USB HID cùng lúc nếu chúng có cùng VID và PID). Các ID này bao gồm tiền tố ("vid_" cho id nhà cung cấp hoặc "pid_" cho id sản phẩm) và 4 chữ số thập lục phân. Ví dụ: MAX3420E có id nhà cung cấp vid_06ba và id sản phẩm pid_5346. Thông thường cả hai giá trị đều có thể được thay đổi trong mã nguồn của thiết bị USB (giả sử bạn có quyền truy cập vào mã này).

[**Kích thước gói / tốc độ truyền dữ liệu trên kết nối**]

Thiết bị HID trao đổi dữ liệu với PC thông qua **báo cáo** HID . Các báo cáo này có thể chứa từ 8 đến 64 byte dữ liệu. Mỗi mili giây, một báo cáo được gửi từ PC tới thiết bị USB và ngược lại. Điều này có nghĩa là tốc độ truyền tối đa theo lý thuyết có thể đạt tới 64.000 byte/giây.

[**Giao diện thư viện USB HID**]

Thư viện được viết bằng **C#** và được phát triển dưới dạng **DLL** , giúp tích hợp vào một dự án mới rất dễ dàng. Thư viện được thiết kế cho các thiết bị USB HID với kích thước 64 byte cho **các điểm cuối** đầu vào và đầu ra . Chỉ cần nhập DLL vào dự án của bạn và truy cập thuận tiện vào các thiết bị USB HID. Tôi đã cố gắng làm cho giao diện phần mềm của thư viện trở nên đơn giản nhất có thể, đồng thời duy trì chức năng tối đa có thể. Hiện tại có sẵn các chức năng sau:

USBInterface(String vid, String pid) - phương thức khởi tạo khởi tạo một thể hiện (biến) mới của lớp. Hàm tạo trong trường hợp này có hai tham số - ID nhà cung cấp và ID sản phẩm của thiết bị USB. Các thông số này sẽ được sử dụng để tìm kiếm thiết bị USB HID cần thiết.

USBInterface(String vid) - phương thức khởi tạo khởi tạo một phiên bản (biến) mới của lớp. Trong trường hợp này, hàm tạo có một tham số - mã định danh nhà cung cấp. Tham số này sẽ được sử dụng để tìm kiếm thiết bị USB HID cần thiết.

Connect() - Phương pháp này thiết lập kết nối với thiết bị USB. Bạn có thể chắc chắn rằng mình chỉ kết nối với đúng thiết bị nếu bạn sử dụng hàm tạo USBInterface với hai tham số - vid và pid. Nếu bạn đã sử dụng hàm tạo với một tham số vid thì kết nối có thể xảy ra với một thiết bị khác có cùng ID nhà cung cấp (VID) - không như bạn mong đợi và bạn sẽ không thể xác định kết nối sẽ được thực hiện với thiết bị nào. Phương thức Connect sẽ trả về false nếu có lỗi xảy ra.

Disconnect() - phương thức ngắt kết nối khỏi thiết bị.

getDeviceList() - phương thức trả về danh sách các thiết bị có ID nhà cung cấp (VID) được chỉ định trong hàm tạo. Bạn sẽ cần chức năng này để tìm hiểu xem có bao nhiêu thiết bị USB được kết nối (và thiết bị nào) với ID nhà cung cấp được chỉ định. Trả về danh sách Chuỗi có đường dẫn hệ thống tới thiết bị.

write(Byte[]) - phương thức ghi một mảng byte vào thiết bị USB. Nếu kích thước mảng vượt quá 64 thì mảng sẽ được chia thành nhiều mảng, mỗi mảng có dung lượng 64 byte. Các mảng sẽ được gửi lần lượt, byte 0..63 sẽ được gửi trước, sau đó byte 64..127 sẽ được gửi, v.v. Phương thức sẽ trả về true nếu tất cả byte được ghi thành công.

startRead() - phương thức được sử dụng để kích hoạt "reading-state" (Trạng thái đọc). Nếu bạn đưa ra lệnh này, nó sẽ bắt đầu một luồng lắng nghe trên thiết bị USB và chờ dữ liệu xuất hiện.

stopRead() - phương thức chuyển thư viện từ "Trạng thái đọc" sang Trạng thái không hoạt động. Bằng cách thực hiện lệnh này, luồng đọc dữ liệu sẽ dừng lại và dữ liệu hiện có thể được truy xuất.

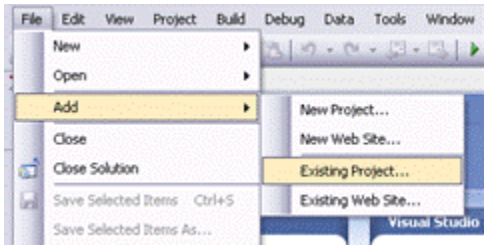
EnableUsbBufferEvent(EventHandler) - Bằng cách gọi phương thức này bằng trình xử lý sự kiện (System.EventHandler), một sự kiện nghe bộ đệm USB sẽ được thêm vào. Bằng cách này, trình xử lý sự kiện sẽ được gọi bất cứ khi nào một tập dữ liệu được thêm vào bộ đệm (và cũng được nhận từ thiết bị USB).

[**Cách tích hợp Thư viện USB HID vào Dự án Visual Basic**]

Có hai cách để tích hợp Thư viện USB HID vào dự án Visual Studio. Một là thêm dự án thư viện vào giải pháp Visual Studio của bạn. Cách thứ hai là thêm tham chiếu đến **USBHIDDRIVER.dll** vào dự án mà bạn muốn sử dụng thư viện USB HID. Trước tiên hãy tải xuống gói thư viện [2] và giải nén các tệp của nó vào dự án ứng dụng của bạn.

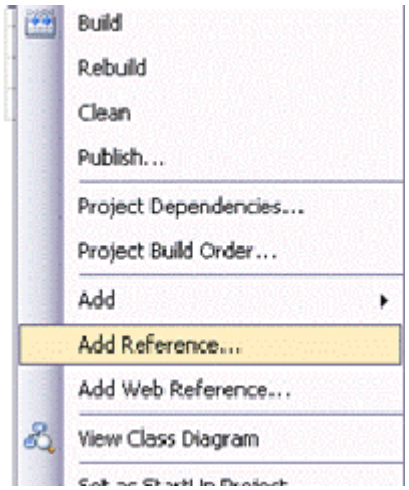
Thêm dự án thư viện vào giải pháp Visual Studio 2005, quy trình từng bước:

1 . Mở giải pháp của bạn (giải pháp Visual Studio). Chọn Tệp -> Thêm -> Dự án hiện có từ menu.

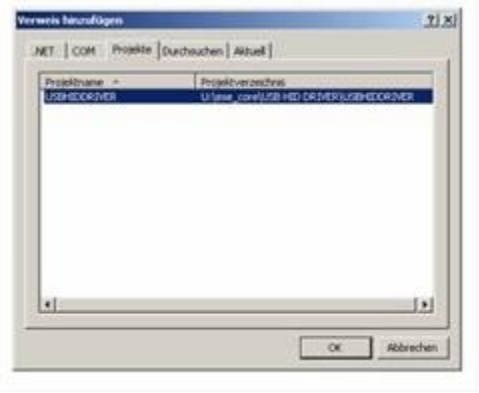


2 . Hộp thoại mở tập tin sẽ mở ra. Tìm tệp USBHIDDRIVER.csproj và mở nó (nhấp vào Mở).

3 . Nhấp chuột phải vào dự án Visual Studio để mở menu ngữ cảnh của dự án và chọn Thêm tài liệu tham khảo.

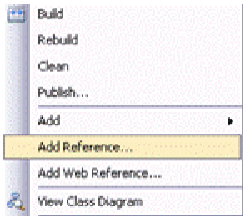


4 . Chọn tab Dự án, sau đó chọn USBHIDDRIVER và nhấp vào OK.



Thêm USBHIDDRIVER.dll vào dự án Visual Studio, quy trình từng bước.

1 . Nhấp chuột phải vào dự án Visual Studio để mở menu ngữ cảnh cho dự án và chọn Thêm tài liệu tham khảo.



2 . Chọn tab Duyệt, tìm USBHIDDRIVER.dll (hoặc USBHIDDRIVER.dll.dll) và nhấp vào OK.

[**Sử dụng USBHIDDRIVER.dll với Visual Basic 6**]

Mã thực thi .Net (thời gian chạy .Net) cho phép các máy khách COM không được quản lý (**các máy khách nhận biết COM không được quản lý** , như các ứng dụng Visual Basic 6) truy cập các thành phần .Net thông qua COM interop (COM interop) và các công cụ do môi trường phát triển cung cấp.

Thư viện USBHIDDRIVER.dll là một tập hợp .NET không thể truy cập được từ Visual Basic 6. Do đó, chúng ta phải tạo một thư viện kiểu có thể được sử dụng với Visual Basic 6. Điều này có thể được thực hiện bằng cách sử dụng công cụ RegAsm.exe, bạn có **thể** tìm trong thư mục cài đặt của môi trường phát triển .Net. Tạo một tệp bó **BAT** trong thư mục chứa USBHIDDRIVER.dll, viết lệnh sau vào đó rồi chạy nó:

```
"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe" USBHIDDRIVER.dll.dll /tlb:
USBHIDDRIVER.dll.tlb
```

Bây giờ bạn phải sao chép cả hai tệp **dll** và **tlb** trong cùng một thư mục vào thư mục ứng dụng, Visual Basic, để nó có thể sử dụng thư viện USB HID.

[**Cách sử dụng Thư viện USB HID**]

Giả sử bạn cần thiết lập trao đổi với thiết bị USB có id nhà cung cấp vid_06ba và một số id sản phẩm có thể có. Điều đầu tiên bạn cần là danh sách tất cả các thiết bị được kết nối với PC có id nhà cung cấp được chỉ định. Để có được danh sách này - chỉ cần tạo một phiên bản mới của lớp USBInterface, đồng thời chỉ định một tham số cho hàm tạo - VID (id nhà cung cấp). Sau đó gọi phương thức getDeviceList và nhận danh sách các thiết bị được kết nối.

```
USBHIDDRIVER. USBInterface usb =  USBInterface mới ( "vid_06ba" ) ;
Chuỗi [ ] list = usbI. getDeviceList ( ) ;
```

Để tổ chức trao đổi với thiết bị mong muốn từ danh sách, hãy tạo một phiên bản USBInterface mới và lần này chỉ định hai tham số cho hàm tạo - VID và PID. Sau đó gọi phương thức Connect và nhận kết nối với thiết bị USB.

```
USBHIDDRIVER. USBInterface usb =  USBInterface mới ( "vid_06ba" , "pid_5346"
) ;
USB. Kết nối ( ) ;
```

Sau khi kết nối với thiết bị được thiết lập, bạn có thể đọc hoặc ghi dữ liệu. Để ghi dữ liệu, chỉ cần gọi phương thức ghi với tham số mảng byte cần truyền. Việc đọc có thể được thực hiện theo hai cách: cách thứ nhất không có sự kiện bộ đệm USB và cách thứ hai với sự kiện bộ đệm USB.

Nếu sự kiện được bật, phương thức xử lý sự kiện sẽ được gọi bất cứ khi nào nhận được một bộ dữ liệu mới. Ngoài ra, bạn có thể xử lý dữ liệu hẹn giờ bằng mã như thế này:

```
USB. viết ( startByte ) ;
USB. EnableUsbBufferEvent ( System. EventHandler mới ( myEventCacher ) ) ;
chủ đề. ngủ ( 5 ) ;
USB. bắt đầuĐọc ( ) ;
…
USB. dừngĐọc ( ) ;
USB. viết ( stopByte ) ;
```

[**Điều gì xảy ra bên trong Thư viện USB HID**]

Thư viện dựa trên lớp **USBSharp** , lớp này nhập tất cả các phương thức cần thiết từ **kernel32.dll** và **hid.dll** . Lớp **HIDUSBDevice** bao bọc các phương thức này và tổ chức việc đọc trong một luồng riêng biệt. Giao diện chính của thư viện là lớp **USBInterface** , có thể truy cập được từ bên ngoài dll. Lớp USBInterface có một đối tượng **ListWithEvent** dựa trên **ArrayList** với phần thưởng là kích hoạt trình xử lý sự kiện khi một bộ dữ liệu mới được thêm vào. Cuối cùng nhưng không kém phần quan trọng, trong lớp **HidApiDeclaration** , bạn sẽ tìm thấy một số khai báo cần thiết để hoạt động với USB HID.

[**Ghi chú của người dịch**]

Trên Internet, bạn có thể tìm thấy một thư viện tương tự như được mô tả trong bài viết này, không chỉ ở dạng DLL mà ở dạng tệp mã nguồn C# (mô-đun .cs) [3]. Thư viện có lớp **UsbLibrary.UsbHidPort** , chứa toàn bộ giao diện để làm việc với thiết bị USB HID. Thư viện này có nhiều tính năng hơn - ví dụ: có các sự kiện rất thuận tiện để kết nối và ngắt kết nối thiết bị USB, giúp dễ dàng thực hiện công việc minh bạch với thiết bị USB HID và khôi phục trao đổi dữ liệu với thiết bị USB mà không cần phải khởi động lại chương trình.

[**Liên kết**]

1 . Trang HID - một trang dành riêng cho lớp HID.
2 . Trình điều khiển / Thư viện USB HID cho Gói thư viện .Net / C# (USBHIDDRIVER.dll) .
3 . Mã nguồn lớp UsbLibrary.UsbHidPort .
4 . Thư viện quản lý thiết bị USB HID .
5 . USB in a NutShell - Hướng dẫn về chuẩn USB .
6 . API HID cho Linux, Mac OS X và Windows .

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using System.Threading;
using Microsoft.Win32.SafeHandles;
using System.IO;

public struct InterfaceDetails
{
        public string manufacturer;
        public string product;
        public string serialNumber;
        public ushort VID;
        public ushort PID;
        public string devicePath;
        public int IN_reportByteLength;
        public int OUT_reportByteLength;
        public ushort versionNumber;
}
public class HIDDevice
{
   #region globals
```

```
    public bool deviceConnected { get; set; }
    public SafeFileHandle handle;
    private HIDP_CAPS capabilities;
    public InterfaceDetails productInfo;
    public byte[] readData;
    #endregion

    #region static_methods

    public static InterfaceDetails[] getConnectedDevices()
    {
                    InterfaceDetails[] devices = new InterfaceDetails[0];

       //Create structs to hold interface information
       SP_DEVINFO_DATA devInfo = new SP_DEVINFO_DATA();
                    devInfo.cbSize = (uint)Marshal.SizeOf(devInfo);
                    SP_DEVICE_INTERFACE_DATA devIface = new SP_DEVICE_INTERFACE_DATA();
       devIface.cbSize = (uint)(Marshal.SizeOf(devIface));

       Guid G = new Guid();
       HID.HidD_GetHidGuid(ref G); //Get the guid of the HID device class

       IntPtr deviceInfo = SetupAPI.SetupDiGetClassDevs(ref G, IntPtr.Zero, IntPtr.Zero,
SetupAPI.DIGCF_DEVICEINTERFACE | SetupAPI.DIGCF_PRESENT);
       //Loop through all available entries in the device list, until false
       int j = 0;
       while (true)
       {
          if (!SetupAPI.SetupDiEnumDeviceInterfaces(deviceInfo, IntPtr.Zero, ref G, (uint)j, ref devIface))
```

```
                {
                        break;
                }
        uint requiredSize = 0;
                IntPtr detailMemory = Marshal.AllocHGlobal((int)requiredSize);
                SP_DEVICE_INTERFACE_DETAIL_DATA functionClassDeviceData =
(SP_DEVICE_INTERFACE_DETAIL_DATA)Marshal.PtrToStructure(detailMemory,
typeof(SP_DEVICE_INTERFACE_DETAIL_DATA));
                functionClassDeviceData.cbSize = Marshal.SizeOf(functionClassDeviceData);
                if (!SetupAPI.SetupDiGetDeviceInterfaceDetail(deviceInfo, ref devIface, ref
functionClassDeviceData, requiredSize, out requiredSize, ref devInfo))
                {
                        Marshal.FreeHGlobal(detailMemory);
                        break;
                }
                string devicePath = functionClassDeviceData.DevicePath;
                Marshal.FreeHGlobal(detailMemory);

                //create file handles using CT_CreateFile
                SafeFileHandle tempHandle = Kernel32.CreateFile(devicePath,
Kernel32.GENERIC_READ | Kernel32.GENERIC_WRITE, Kernel32.FILE_SHARE_READ |
Kernel32.FILE_SHARE_WRITE,
          IntPtr.Zero, Kernel32.OPEN_EXISTING, 0, IntPtr.Zero);

        //get capabilites - use getPreParsedData, and getCaps
        //store the reportlengths
        IntPtr ptrToPreParsedData = new IntPtr();
        bool ppdSucsess = HID.HidD_GetPreparsedData(tempHandle, ref ptrToPreParsedData);
        if (!ppdSucsess)
                {
```

```
                continue;
            }

HIDP_CAPS capabilities = new HIDP_CAPS();
HID.HidP_GetCaps(ptrToPreParsedData, ref capabilities);

HIDD_ATTRIBUTES attributes = new HIDD_ATTRIBUTES();
HID.HidD_GetAttributes(tempHandle, ref attributes);

string productName = EMPTY_STRING;
string SN = EMPTY_STRING;
string manfString = EMPTY_STRING;
            const int bufferLen = 128;
IntPtr buffer = Marshal.AllocHGlobal(bufferLen);
if (HID.HidD_GetProductString(tempHandle, buffer, bufferLen))
            {
                productName = Marshal.PtrToStringAuto(buffer);
            }
            if (HID.HidD_GetSerialNumberString(tempHandle, buffer, bufferLen))
            {
                SN = Marshal.PtrToStringAuto(buffer);
            }
            if (HID.HidD_GetManufacturerString(tempHandle, buffer, bufferLen))
            {
                manfString = Marshal.PtrToStringAuto(buffer);
            }
Marshal.FreeHGlobal(buffer);

            //Call freePreParsedData to release some stuff
```

```
                        HID.HidD_FreePreparsedData(ref ptrToPreParsedData);

                        //If connection was sucsessful, record the values in a global struct
                        InterfaceDetails productInfo = new InterfaceDetails();
        productInfo.devicePath = devicePath;
        productInfo.manufacturer = manfString;
        productInfo.product = productName;
        productInfo.PID = (ushort)attributes.ProductID;
        productInfo.VID = (ushort)attributes.VendorID;
        productInfo.versionNumber = (ushort)attributes.VersionNumber;
        productInfo.IN_reportByteLength = (int)capabilities.InputReportByteLength;
        productInfo.OUT_reportByteLength = (int)capabilities.OutputReportByteLength;
        productInfo.serialNumber = SN;     //Check that serial number is actually a number

        int newSize = devices.Length + 1;
        Array.Resize(ref devices, newSize);
        devices[newSize - 1] = productInfo;
                        ++j;
                }
    SetupAPI.SetupDiDestroyDeviceInfoList(deviceInfo);

    return devices;
}

#endregion

#region constructors
/// <summary>
/// Creates an object to handle read/write functionality for a USB HID device
```

```
    /// Uses one filestream for each of read/write to allow for a write to occur during a blocking
    /// asnychronous read
    /// </summary>
    /// <param name="VID">The vendor ID of the USB device to connect to</param>
    /// <param name="PID">The product ID of the USB device to connect to</param>
    /// <param name="serialNumber">The serial number of the USB device to connect to</param>
    /// <param name="useAsyncReads">True - Read the device and generate events on data being
available</param>
    public HIDDevice(ushort VID, ushort PID)
    {
                    InterfaceDetails[] devices = getConnectedDevices();

       //loop through all connected devices to find one with the correct details
       for (int i = 0; i < devices.Length; i++)
       {
          if ((devices[i].VID == VID) && (devices[i].PID == PID))
                                {
                                        initDevice(devices[i].devicePath);
                                        break;
                                }
       }
    }

    /// <summary>
    /// Creates an object to handle read/write functionality for a USB HID device
    /// Uses one filestream for each of read/write to allow for a write to occur during a blocking
    /// asnychronous read
    /// </summary>
    /// <param name="devicePath">The USB device path - from getConnectedDevices</param>
```

```
   /// <param name="useAsyncReads">True - Read the device and generate events on data being available</param>
   public HIDDevice(string devicePath)
   {
      initDevice(devicePath);
   }
   #endregion
   private void initDevice(string devicePath)
   {
      deviceConnected = false;

                  //create file handles using CT_CreateFile
                  handle = Kernel32.CreateFile(devicePath, Kernel32.GENERIC_READ | Kernel32.GENERIC_WRITE, Kernel32.FILE_SHARE_READ | Kernel32.FILE_SHARE_WRITE, IntPtr.Zero, Kernel32.OPEN_EXISTING, 0, IntPtr.Zero);

      //get capabilites - use getPreParsedData, and getCaps
      //store the reportlengths
      IntPtr ptrToPreParsedData = new IntPtr();
      HID.HidD_GetPreparsedData(handle, ref ptrToPreParsedData);

      capabilities = new HIDP_CAPS();
      HID.HidP_GetCaps(ptrToPreParsedData, ref capabilities);

      HIDD_ATTRIBUTES attributes = new HIDD_ATTRIBUTES();
      HID.HidD_GetAttributes(handle, ref attributes);

      string productName = EMPTY_STRING;
      string SN = EMPTY_STRING;
      string manfString = EMPTY_STRING;
```

```
IntPtr buffer = Marshal.AllocHGlobal(126);//max alloc for string;
if (HID.HidD_GetProductString(handle, buffer, 126))
                {
                        productName = Marshal.PtrToStringAuto(buffer);
                }
                if (HID.HidD_GetSerialNumberString(handle, buffer, 126))
                {
                        SN = Marshal.PtrToStringAuto(buffer);
                }
                if (HID.HidD_GetManufacturerString(handle, buffer, 126))
                {
                        manfString = Marshal.PtrToStringAuto(buffer);
                }
Marshal.FreeHGlobal(buffer);

                //Call freePreParsedData to release some stuff
                HID.HidD_FreePreparsedData(ref ptrToPreParsedData);

if (handle.IsInvalid)
   return;

deviceConnected = true;

//If connection was sucsessful, record the values in a global struct
productInfo = new InterfaceDetails();
productInfo.devicePath = devicePath;
productInfo.manufacturer = manfString;
productInfo.product = productName;
productInfo.serialNumber = SN;
```

```
    productInfo.PID = (ushort)attributes.ProductID;
    productInfo.VID = (ushort)attributes.VendorID;
    productInfo.versionNumber = (ushort)attributes.VersionNumber;
    productInfo.IN_reportByteLength = (int)capabilities.InputReportByteLength;
    productInfo.OUT_reportByteLength = (int)capabilities.OutputReportByteLength;
  }
  public void close()
        {
                try
                {
                        if (handle != null && !handle.IsInvalid)
                        {
                                //Kernel32.PurgeComm(handle, Kernel32.PURGE_TXABORT |
Kernel32.PURGE_RXABORT);
                                //Kernel32.CloseHandle(handle);
                                handle.Close();
                                handle = null;
                                logInfo("设备已关闭");
                        }
                        this.deviceConnected = false;
                }
                catch(Exception e)
                {
                        logInfo("exception : " + e.Message);
                }
  }
  public bool write(byte[] data)
  {
    if (data.Length >= capabilities.OutputReportByteLength)
```

```
                {
                        return false;
                }
                uint writeCount = 0;
                return Kernel32.WriteFile(handle, data, (uint)data.Length, ref writeCount, IntPtr.Zero);
        }
    public int read(ref byte[] data, int expectCount = 0)
    {
                if (data.Length < capabilities.InputReportByteLength)
                {
                        return 0;
                }
                uint readCount = 0;
                Kernel32.ReadFile(handle, data, (uint)data.Length, ref readCount, IntPtr.Zero);
                // 如果读取的数量和期望的数量不一致,则丢弃数据
                if(expectCount > 0 && expectCount != readCount)
                {
                        return 0;
                }
                return (int)readCount;
        }
}
```