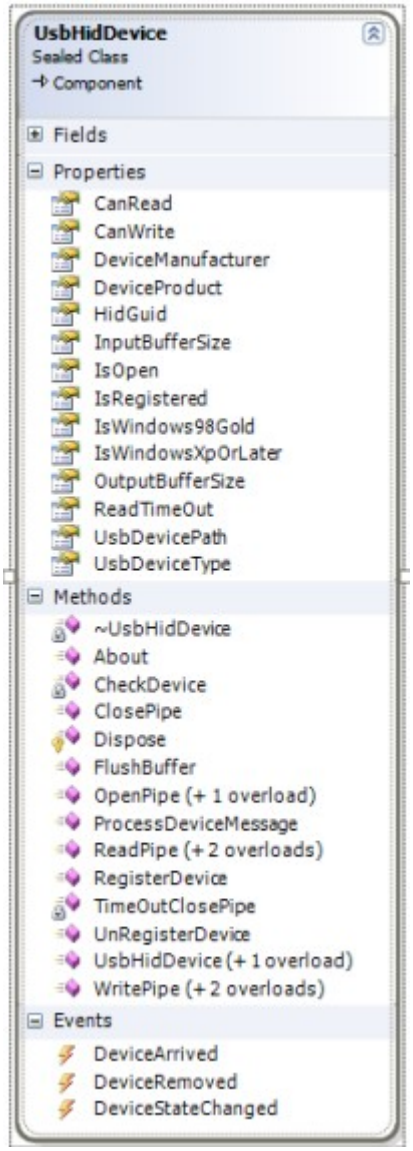
USBManagement ve PIC Firmware Açıklamaları

USBManagement Library – umng.dll

Bu DLL içinde HID sınıfı cihazlar ile haberleşmek, cihaz bilgilerini almak ve cihaz takibi için gerekli methodlar ve özellikler bulunmaktadır.Şimdi bu özelliklerin ve methodların uygulamalarda nasıl kullanılacağını inceleyelim.



ÖZELLİKLER – Properties

CanRead – CanWrite : Bu özellikler cihaz ile bağlantı açılmaya çalışıldıktan sonra yazma ve okuma için handle alma işleminin başarısını gösterirler. Her iki özellikte işlemler başarılı ise TRUE değerini alır.

DeviceManufacturer : Bu özellik cihaz firmware'ında tanımlanmış üretici string'ini gösterir.

DeviceProduct : Bu özellik cihaz firmware'ında tanımlanmış ürün string'ini gösterir.

HidGuid : Bu özellik HID Sınıfı'nın GUID numarasını döndürür.

InputBufferSize : Bu özellik cihaz ile bağlantı kurulduktan sonra güncellenir ve HID Rapor tanımlayıcısında belirtilen InputBufferSize değerini içerir. PC bu özellikten okunan uzunluğun bir eksiği kadar veri gönderilebilir.

IsOpen : Bu özellik cihaz ile PC arasındaki bağlantı durumunu gösterir. Bağlantı açık ise TRUE aksi halde FALSE'dur.

IsRegistered : Bu özellik cihaz takibi için, cihazın sisteme kayıt edilip edilmediğini gösterir.Cihaz RegisterDevice() methodu ile sisteme kaydedildikten sonra bu özellik TRUE olur.UnRegisterDevice() ile sistemden kaldırıldığında ise FALSE olur.Cihaz takibine bu methodlar anlatılırken değinilecektir.

IsWindows98Gold : Bu özellik işletim sistemi 98-Gold ise TRUE aksi halde FALSE döndürür.98-Gold'da kesme OUT transfer'ler mümkün olmadığından bu özellik kontrol edilip transfer işlemi ona göre belirlenmelidir.Eğer sistem 98-Gold ise OUT transfer "Kontrol Transfer" kullanılarak kotarılmalıdır.

IsWindowsXpOrLater : Bu özellik IsWindows98Gold özelliğindeki işlem için kullanılabilir.Tek farkı sistem versiyonunun XP veya sonrası bir sistem olduğuna bakmasıdır.

OutputBufferSize : Bu özellik cihaz ile bağlantı kurulduktan sonra güncellenir ve HID Rapor tanımlayıcısında belirtilen OutputBufferSize değerini içerir.Cihaz'a PC'den ancak bu özellikten okunan uzunluğun bir eksiği kadar bilgi gönderilebilir.

ReadTimeOut : Bu özellik okuma zaman aşımı süresini tutar veya ayarlar.Varsayılan zaman aşımı süresi 3 saniyedir.Okuma işlemi bu süre içinde gerçekleşmezse ortama bir ReadTimeOutException ististanısı fırlatılır. Bu istisna ve diğerlerine methodlar incelenirken değinilecektir.
ReadTimeOut süresi istenilirse milisaniye cinsinde değiştirilebilinir.Örneğin bu özellik 3 saniye için 3000 değerini taşır.

UsbDevicePath : Bu özellik istenen cihaz bulunduktan ve bağlantı kurulduktan sonra güncellenir ve cihazın user moddan erişebilmesi için sembolik path ismini tutar.

UsbDeviceType : Bu özellik DeviceType numaralandırıcısından herhangi birini alır.Bağlantı kurulan cihaz'ın tipi kontrol edilir ve bu özellik güncellenir.Bu özellik aşağıdaki tipleri içerebilir;

Unknown : Cihaz tipi bilinmiyor(Kullanıcıya ait HID arayüzlü özel cihaz)
Keyboard : Cihaz bir klavye
Mouse: Cihaz bir mouse

METHODLAR – Methods

```
public void About(IntPtr ApplicationHandle)
```

Açıklama:
Bu method USBManagament ile ilgili tanıtıcı bir about ekranı açar.

Parametreler:
ApplicationHandle:
Bu parametreye ana uygulama form'unun handle değeri verilmelidir.

Örnek:
```
UsbHidDevice device = new UsbHidDevice();
device.About(this.Handle);
```

Dönen değer:
Bu method herhangi bir değer döndürmez.

```
public bool ClosePipe()
```

Açıklama:
Bu method daha önce açık olan bir bağlantıyı kapatır ve ardından DeviceStateChanged olayını tetikler.

Parametreler:
UsbHidDevice örneği varsayılan bağlantıyı saklar.Bu yüzden herhangi bir parametre almadan sakladığı bağlantıyı kapatır.

Exception:
Eğer daha önceden açılmış bir bağlantı yoksa bu methodu çağırmak bir AllReadyCloseException istisnası oluşturur.Bu istisnayı almamak için bağlantı ilk önce IsOpen özelliği ile kontrol edilmeli ardından kapatılmalıdır

Örnek:
```
UsbHidDevice device = new UsbHidDevice();
device.DeviceStateChanged += new DeviceStateChangeEventHandler(OnChangeDeviceState);

if(device.IsOpen)
{
   if(device.ClosePipe()) MessageBox.Show("Bağlantı kapatıldı");
   else MessageBox.Show("Bağlantı kapatma işlemi başarısız");
}

private void OnChangeDeviceState(object sender, DeviceStateChangeEventArgs e)
{
   if(e.CurrentState == DeviceState.Closed)
   {
      //Bağlantı kapandıktan sonra çalıştırılmak istenen kodlar
   }
}
```
Dönen Değer:
Eğer bağlantı başarılı bir şekilde kapatılırsa TRUE aksi halde FALSE döndürür.

```
public bool FlushBuffer()
```

Açıklama:
Bu method buffer'da bekleyen input raporunu temizler.

Parametreler:
Bu method herhangi bir parametre almaz.

Exception:
Bu method çağrıldığında eğer cihaz ile bağlantı yok ise bir DeviceNotConnectException istisnası, eğer buffer temizleme işlemi geçersiz bir handle değeri ile yürütülmeye kalkılırsada InvalidHandleException istisnası fırlatılır.

Dönen Değer:
Tampon temizleme işlemi başarılı ise TRUE aksi halde FALSE döner.

```
public bool OpenPipe(short VendorId, short ProductId)
```

Açıklama:
Bir aşırı yüklenmiş versiyonu bulunan bu methodun, bu ilk versiyonu VendorId ve ProductId'si verilen cihazı sistemde arar ve bulduğunda diğer tüm işlemlerin yapılabilmesi için bağlantı kurar.Bu durumda IsOpen değeri TRUE olur ve DeviceStateChanged olayını tetikler.

Parametreler:
VendorId:
Cihaz firmware'ı tarafından tanımlanan VendorId değeri bu parametreye geçirilmelidir.

ProductId:
Cihaz firmware'ı tarafından tanımlanan ProductId değeri bu parametreye geçirilmelidir.

Exception:
Eğer cihaz ile önceden kurulmuş bir bağlantı var ise bu methodu tekrar çağırmak bir AlReadyDeviceException istisnası fırlatılmasına neden olur.

Örnek:

```
UsbHidDevice device = new UsbHidDevice();
device.DeviceStateChanged += new DeviceStateChangeEventHandler(OnChangeDeviceState);

if(!device.IsOpen)
{
    if(device.OpenPipe(vid, pid)) MessageBox.Show("Bağlantı açıldı");
    else MessageBox.Show("Bağlantı açılamadı");
}

private void OnChangeDeviceState(object sender, DeviceStateChangeEventArgs e)
{
    if(e.CurrentState == DeviceState.Opened)
    {
        //Bağlantı açıldıktan sonra çalıştırılmak istenen kodlar
    }
}
```

Dönen değer:
Eğer bağlantı başarılı bir şekilde açıldıysa TRUE aksi halde FALSE döner.

```
public bool OpenPipe(short VendorId, short ProductId, int timeOut)
```

Açıklama:
Bu method'un önceki versiyonundan tek farkı, milisaniye cinsinden belirtilen zaman aşımı süresi sonunda açılan bağlantının otomatik olarak kapatılmasıdır.ReadTimeOut süresi bu değer ile aynı olmamalıdır ve daima ReadTimeOut süresi daha küçük olmalıdır.

```
public void ProcessDeviceMessage(Message m)
```

Açıklama:
Bu method önceden RegisterDevice() methodu ile sisteme kaydedilmiş bir cihazı izlemek amacıyla pencere fonksiyonu tarafından çağrılır.Bir programın pencere fonksiyonu işletim sisteminin uygulamanın mesaj kuyruğuna bıraktığı mesajları işler.Bu fonksiyon içinden sürekli bu method çağrılarak WM_DEVICECHANGE olayı izlenir ve takılan ya da sökülen cihaz sisteme kayıtlı cihaz ise DeviceArrived ve DeviceRemoved olayları tetiklenerek kullanıcı durumdan haberdar edilir.Ana uygulamanın pencere fonksiyonu WndProc'dur ve program içinde override edilmelidir.

Parametreler:
m:
Bu paramtreye işletim sistemi tarafından doldurulan ve WndProc pencere fonksiyonuna geçirilen Message tipindeki yapı geçirilmelidir.

Örnek:
```
UsbHidDevice device = new UsbHidDevice();

device.DeviceArrived += new HidDeviceArrivedEventHandler(OnDeviceAttached);
device.DeviceRemoved += new HidDeviceRemovedEventHandler(OnDeviceDetached);

if(device.OpenPipe(vid, pid))
    if(!device.IsRegistered) device.RegisterDevice(this.Handle);

private void OnDeviceAttached(object sender, EventArgs e)
{
    if(!device.IsOpen) device.OpenPipe(vid, pid);
    //Cihaz algılandığında işlenecek diğer kodlar
}

private void OnDeviceDetached(object sender, EventArgs e)
{
    if(device.IsOpen) device.ClosePipe();
    //Cihaz söküldüğünde işlenecek diğer kodlar
}

protected override void WndProc(ref Message m)
{
    device.ProcessDeviceMessage(m);
    base.WndProc(ref m);
}
```

Dönen Değer:
Bu method herhangi bir değer döndürmez.

```
public bool ReadPipe(ref byte[] buffer, int Length, TransactionType transaction)
```

Açıklama:
İki aşırı yüklenmiş versiyonu bulunan bu methodun, bu ilk versiyonu cihazdan TransactionType ile belirlenmiş transfer türünü kullanarak referans olarak verilen buffer bölgesine Length kadar veri okur.Burada ki Length ne kadar uzunlukta verilirse verilsin method tarafından otomatik olarak InputBufferSize değerine çekilir.Bu yüzden uzunluk HID Rapor tanımlayıcısı tarafından belirlenir.Okuma işlemi iki şekilde sonlanabilir.Birincisinde cihazdan veriler başarı bir şekilde okunmuştur, ikincisinde ise ReadTimeOut ile belirtilen süre dolmuş ve hiç veri okunamamıştır.Bu durumda ortama bir ReadTimeOutException istisnası fırlatılır.

Parametreler:
buffer:
Verilerin okunacağı buffer alanı.Bir byte dizisi alan bu parametreye, bu dizinin referansı geçirilmelidir.

Length:
Okunacak verilerin uzunluğun gösterir.Bu değer InputBufferSize'a eşit olmalıdır.

transaction:
Bu değer transferin türünü belirler ve TransactionType numaralandırıcısı ile belirtilen değerlerden birini alabilir.Bu değerler Control ya da Interrupt olabilir.Bu durumda transfer ya kontrol transfer ile ya da kesme transfer ile gerçekleştirilir.

Exception:
Eğer mevcut bir bağlantı yokken okuma işlemi yapılmaya çalışılırsa ortama bir DeviceNotConnectException istisnası fırlatılır.Aynı zamanda dahili olarak geçersiz bir okuma handle'ına yapılan çağrı InvalidHandleException istisnasına yol açar.Bu yüzden okuma yapılmadan önce bağlantı kontrol edilmeli ve CanRead özeliği dikkate alınarak okuma işlemi yapılmalıdır.

Örnek:
```
  UsbHidDevice device = new UsbHidDevice();

  byte[] buffer;
  device.ReadTimeOut = 5000;   //5 saniye içinde veri alımazsa fonksiyon bitecek
                               //ve istisna fırlatılacak.

  if(!device.IsOpen)
  {
     if(device.OpenPipe(vid, pid))
     {
        try
        {
            if(device.CanRead)
            {
               buffer = new byte[device.InputBufferSize];
               if(device.ReadPipe(ref buffer, buffer.Length, TransactionType.Interrupt))
                  MessageBox.Show("Veriler okundu");
               else
                  MessageBox.Show("Veriler okunamadı");
            }
         }
         catch(ReadTimeOutException err)
         {
            //Zaman aşımı
            MessageBox.Show(err.Message);
         }
      }
   }
```

Dönen Değer:
Eğer okuma işlemi başarılı ise TRUE, aksi halde FALSE döner.

```
public bool ReadPipe(ref byte[] buffer, int Length, TransactionType transaction, bool
autoClose)
```

Açıklama:
Bu method bir önceki anlatılan ReadPipe()'ın aşırı yüklenmiş versiyonudur ve okuma işlemi sonrası bağlantının otomatik kapatılmasını sağlaması dışında herşey aynıdır.Eğer autoClose TRUE verilirse okuma işlemi sonrası bağlantı kapatılır.

```
public bool ReadPipe(ref byte[] buffer, int Length, TransactionType transaction,
AutoResetEvent are, bool autoClose)
```

Açıklama:
Bu method önceki ReadPipe()'lar gibi çalışır ancak diğerlerinden tek farkı okuma işlemi sonrası bir AutoResetEvent nesnesini set'lemesidir.Bu çok kanallı çalışmalarda (multithreading) faydalıdır.Örneğin USB cihaz paylaşılan bir kaynak ise ve herhangi bir zamanda herhangi bir thread bu cihazdan okuma yapıyorsa diğer thread'lerin cihaza erişmemesi istenir.Bu işlem okuma yapan blok, lock ile kitlenerek halledilebilinir.Fakat diğer kanalların bir olay nesnesini beklemesi ve set'lendiğinde sıradaki thread'in haberdar olması daha sağlam bir çözümdür.

Örnek:

```
AutoResetEvent are = new AutoResetEvent();
UsbHidDevice device = new UsbHidDevice();

device.ReadPipe(ref buffer, buffer.Length, TransactionType.Interrupt, are, false);
are.WaitOne();

MessageBox.Show("Okuma işlemi bitti");
```

```
public bool RegisterDevice(IntPtr FormHandle)
```

Açıklama:
Bu method ProcessDeviceMessage() ile cihazın izlenmesi için, cihazı sisteme kaydeder.

Parametreler:
FormHandle:
Bu paramtreye ana uygulamanın (main form) handle değeri geçirilir.Böylece kaydedilen cihaz hakkındaki değişikliklerin iletileceği form belirlenmiş olunur.

Exception:
Cihaz daha önce sisteme kaydedilmişse bu methodu tekrar çağırmak ortama bir AlReadyRegisteredException istisnası fırlatılmasına neden olur.Aynı zamanda daha önce bağlantı kurulmamış bir cihaz'ı sisteme kaydetmeye çalışmak bir DeviceNotConnectException fırlatılmasına neden olur.Bu yüzden IsOpen ve IsRegistered özellikleri kontrol edilmeden bu method çağrılmamalıdır.

Örnek:

```
UsbHidDevice device = new UsbHidDevice();

if(!device.IsOpen)
{
   if(device.OpenPipe(vid, pid))
   {
      if(!device.IsRegistered) device.RegisterDevice(this.Handle);
   }
}
```

Dönen Değer:
Cihaz sisteme başarılı bir şekilde kaydedilirse TRUE, aksi halde FALSE döndürülür.

```
public bool UnRegisterDevice()
```

Açıklama:
Bu method daha önce sisteme RegisterDevice() ile kaydedilmiş bir cihazı sistemden kaldırır.

```
public bool WritePipe(byte[] buffer, TransactionType transaction)
```

Açıklama:
Bu methodunda iki aşırı yüklenmiş versiyonu mevcuttur.Bu ilk versiyon, buffer ile belirtilen alandaki verileri TransactionType ile belirtilen transfer türünü kullanarak cihaz'a gönderir.Dikkat edilirse buffer için bir uzunluk belirlemeye gerek yoktur.Buffer büyüklüğü ne kadar büyük olursa olsun WritePipe bu tamponu cihazın Rapor tanımlayıcısında belirtilmiş OutputBufferSize'a çeker.

Parametreler:
buffer:
Bu parametreye cihaza gönderilecek verilerin bulunduğu bir byte dizisi geçilir.

TransactionType:
Bu parametreye veriler gönderilirken kullanılacak transfer türü geçilir.

Exception:
Eğer cihaz ile mevcut bir bağlantı yokken yazma girişiminde bulunmak DeviceNotConnectException istisnasının fırlatılmasına neden olur.Aynı zamanda geçersiz bir handle değeri ile yazmaya çalışmak InvalidHandleValueException istisnasının fırlatılmasına neden olur.Bu yüzden ilk önce bağlantı kontrol edilmeli ardından CanWrite ile kontrol yapılıp öyle WritePipe çağrılmalıdır.

Örnek:
```
  UsbHidDevice device = new UsbHidDevice();

  byte[] buffer;

  if(!device.IsOpen)
  {
     if(device.OpenPipe(vid, pid))
     {
        if(device.CanWrite)
        {
           buffer = new byte[device.OutputBufferSize];
           for(int i = 0; i < buffer.Length; i++) buffer[i] = i;

           if(device.WritePipe(buffer, TransactionType.Interrupt))
              MessageBox.Show("Veriler gönderildi");
           else
              MessageBox.Show("Veriler gönderilemedi");
        }
      }
   }
```

Dönen Değer:
Yazma işlemi başarılı ise TRUE, aksi halde FALSE döner.

Bu methodun diğer aşırı yüklü versiyonları aynı ReadPipe() daki mantıktadır.AutoClose özelliği ile yazma işlemi sonrası bağlantı kapatılır, AutoResetEvent nesnesi ile de yazma sonrası bu olay nesnesi setlenir.

USB PIC Firmware
Bu proje içerisinde USB transferleri, listelemeyi ve kontrol işlemlerini yürütecek birkaç kod dosyası ve başlık dosyası bulunur.Bunların işlevleri ve içerisindeki kodlar ileride ayrıntılı olarak anlatılacaktır.
Fakat USB ile uygulama geliştirirken kullanıcıların ilgilenmesi gereken asıl dosyalar devProc.c ve devProc.h dosyalarıdır.Bu dosyalar DeviceProcedure'lerini içerir.Kullanıcı asıl yapması gereken işlemleri devProc.c'ye tanımlamalarını ise devProc.h'a yapar.devProc.c'de iki temel fonksiyon tanımlıdır;

```
void UserInit( void ) {
}
```

Bu fonksiyon gövdesine kullanıcı istediği başlangıç ayarlarını kodlar.Örneğin USB'den sıcaklık bilgisi okunacak bir projede bu fonksiyon kapsamına ADC ayarları, Port ayarları gibi kodlar eklenir.

```
void UserProc(unsigned char Direction)
{
    if(Direction == INTERRUPT_OUT)
    {
       HidTransferReadService(Buffer, HID_OUT_ENDPT_SIZE);
       //Diğer Kodlar
    }
    else if(Direction == INTERRUPT_IN)
    {
       if(!EndPoint1BdtIn.BDnSTAT.UOWN) HidTransferWriteService(
                                        Buffer,
                                        HID_IN_ENDPT_SIZE
                                        );

       //Diğer Kodlar
    }
}
```

Bir diğer fonksiyon ise UserProc olan Kullanıcı Prosedürüdür.Bu kapsam içine asıl transfer kodları yerleştirilir.

Yukarıdaki kod örneğinde belirtilmiş yapı iskelet yapıdır ve değiştirilmemelidir.UserProc Direction adında bir parametre alır.Bu parametre transferin yönünü gösteren INTERRUPT_OUT ve INTERRUPT_IN adında iki sabit alır.INTERRUPT_OUT PC'den cihaza veri geldiğini, INTERRUPT_IN ise PC'nin cihazdan veri göndermesini istedeğini belirtir.INTERRUPT_IN PC'nin cihazı yoklaması sırasında sürekli oluştuğundan bu olay her oluştuğunda gönderilecek veri olsada olmasada gönderme işlemi yapılmalıdır.umng.dll içindeki ReadPipe methodu aslında bu fonksiyonu çalıştırmaz.Bu method HID tamponundaki en son ki yoklama sonrasındaki veriyi okur.WritePipe methodu ise INTERRUPT_OUT'u çalıştırır.Bu durumda da transferin düzgün işlemesi için verilerin okunması gerekir.Yukarıda okuma ve gönderme işlemlerini `HidTransferReadService` ve `HidTransferWriteService` fonksiyonları yapar.Yukarıdaki yapı değiştirilmeden hr türlü kodlama yapılabilir. Aşağıda USBMotorControl'ün örnek kodu verilmiştir.

```
unsigned char PwmBuffer[HID_IN_ENDPT_SIZE];

void UserProc( unsigned char Direction )
{
	if((bDeviceState < CONFIGURED)||(UCONbits.SUSPND == true)) return;
	if(Direction == INTERRUPT_OUT)
	{
		HidTransferReadService(PwmBuffer, HID_OUT_ENDPT_SIZE);
		if(PwmBuffer[CMD_OPEN_CLOSE] == OPEN_MOTOR) { mStartPwmOfTimer(); }
		else if(PwmBuffer[CMD_OPEN_CLOSE] == CLOSE_MOTOR) { mStopPwmOfTimer(); }
		else
		{
			CCPR2L = PwmBuffer[0];
			CCP2CON |= (PwmBuffer[1] << 4);
		}
	}
	else if(Direction == INTERRUPT_IN)
	{
		if(!EndPoint1BdtIn.BDnSTAT.UOWN) HidTransferWriteService(PwmBuffer,
HID_IN_ENDPT_SIZE);
	}
}
```

Her kullanıcı kendine ait buffer alanını belirler.Tanımlama işlemi örnek kodun en üstündeki gibi olmalıdır.Bu standart uzunluk olan 4 byte'ı tanımlar.
Tampon'un genişletilebilmesi için Rapor Tanımlayıcısında değişiklikler yapılması gerekir.Bu konu ilerki bölümlerde ele alınacaktır.

JaBBa