

SpacePoint Fusion Game Pad Application Note

Introduction

PNI's SpacePoint Fusion module streams raw accelerometer, gyroscope, and magnetometer data, as well as calculated orientation information represented as quaternions, through a USB/HID interface. No custom drivers are needed.

This application note describes the data streaming from the SpacePoint Fusion module, explains how to use the [SpacePoint Game Pad](#) application to display or log sensor and calculated quaternion data, and provides sample source code for the SpacePoint Game Pad application. You may want to skip ahead to the "Running SpacePoint Game Pad" section if you simply want to begin logging SpacePoint Fusion data.

Interpreting SpacePoint Fusion's Streaming Output

SpacePoint Fusion is a USB composite device with two interfaces. The first interface contains endpoint 0x81 and the second interface contains endpoint 0x82. HID Pointer Report is sent on endpoint 0x81, and HID Game Pad Report is sent on endpoint 0x82. The HID Pointer Report includes 20 bytes of data, and the Game Pad Report includes 15 bytes of data. For additional information regarding USB, refer to the [USB 2.0 specification](#) and the [USB HID specification](#).

Output data on endpoint 0x81 includes raw sensor data sampled on the magnetic sensors, accelerometers and gyroscopes. Output on endpoint 0x82 includes scaled acceleration values converted into g's and orientation values given as quaternions. These are calculated from the sensor data using PNI's proprietary Spacepoint algorithm, which is embedded in the Spacepoint firmware. Button status and PNI reserved data fields also are reported at both endpoints. The data is updated and output at 125 Hz, except for the quaternions and scaled acceleration values which are updated at 62.5 Hz and output at 125 Hz. All data is in hex format, and data for each sensor axis is 2 bytes in little Endian format.

The 20 bytes of data on endpoint 0x81 is presented in little Endian format as follows:

Byte	Description
0	Mag X Lower
1	Mag X Upper
2	Mag Y Lower
3	Mag Y Upper
4	Mag Z Lower
5	Mag Z Upper
6	Accel X Lower
7	Accel X Upper
8	Accel Y Lower
9	Accel Y Upper
10	Accel Z Lower
11	Accel Z Upper
12	Gyro Roll Lower
13	Gyro Roll Upper
14	Gyro Pitch Lower
15	Gyro Pitch Upper
16	Gyro Yaw Lower
17	Gyro Yaw Upper
18	Reserved/Buttons
19	PNI Reserved

The status of the SpacePoint Fusion's left and right buttons are packed into the 18th byte. The least significant bit (LSB) is the left button and the second LSB is the right button.

The SpacePoint Fusion adheres to the North-East Down (NED) orientation system.

A sample data string is given below, along with the conversion to decimal.

Endpoint 0x81 data stream: 0181fd80fa805780ae801e81fd7d007e067ee065

Raw	01	81	fd	80	fa	80	57	80	ae	80	1e	81	fd	7d	00	7e	06	7e	e0	65
Desc	Mag X		Mag Y		Mag Z		Accel X		Accel Y		Accel Z		Gyro R		Gyro P		Gyro Y		Res/B0/B1	PNI Res
Hex	8101		80fd		80fa		8057		80ae		811e		7dfd		7e00		7e06		e0	65
Dec	33025		33021		33018		32855		32942		33054		32253		32256		32262		-	-

The 15 bytes of data on endpoint 0x82 is presented in little Endian format as follows:

Byte	Description
0	Scaled Accel X Lower
1	Scaled Accel X Upper
2	Scaled Accel Y Lower
3	Scaled Accel Y Upper
4	Scaled Accel Z Lower
5	Scaled Accel Z Upper
6	Quaternion [0] Lower
7	Quaternion [0] Upper
8	Quaternion [1] Lower
9	Quaternion [1] Upper
10	Quaternion [2] Lower
11	Quaternion [2] Upper
12	Quaternion [3] Lower (scalar)
13	Quaternion [3] Upper (scalar)
14	Reserved/Button

The status of the SpacePoint Fusion's left and right buttons are packed into the 14th byte. The least significant bit (LSB) is the left button and the second LSB is the right button.

Endpoint 0x82 data: 2f85238a5b90ac9f496a126a1ef8d0

Raw	2f	85	23	8a	5b	90	ac	9f	49	6a	12	6a	1e	f8	d0
Desc	Scaled AccelX		Scaled AccelY		Scaled AccelZ		q[0]		q[1]		q[2]		q[3]		Res/B0/B1
Hex	852f		8a23		905b		9fac		6a49		6a12		f81e		d0
Dec	34095		35363		36955		40876		27209		27154		63518		-
Offset	32768		32768		32768		32768		32768		32768		32768		-
Counts	1327		2595		4187		8108		-5559		-5614		30750		-
Scale Factor	6		6		6		1		1		1		1		-
Scaled Counts	7962		15570		25122		8108		-5559		-5614		30750		-
Normal. Factor	32768		32768		32768		32768		32768		32768		32768		-
Value	0.2430		0.4752		0.7667		0.2474		-0.1697		-0.1713		0.9384		-

Where:

- "Offset" = the middle of the sensor's total range, or 32768.
- "Counts" = "Dec" - "Offset"
- "Scale Factor" = the 'g' rating of the accelerometer
- "Scaled Counts" = "Counts" * "Scale Factor"
- "Normal. Factor" = the normalization factor for the sensor, or 32768
- "Value" = "Scaled Counts" / "Normalization Factor"

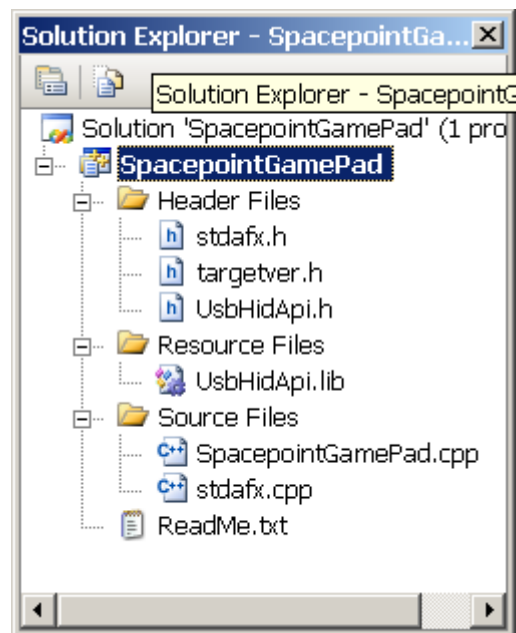
Overview of SpacePoint Game Pad

The [SpacePoint Game Pad zip file](#) includes a complete Visual C++ project, with a functioning build in the “Release” folder. (The “Release” folder is created when the zip file is extracted.) The executable file in the “Release” folder can be used to display or log sensor or calculated data from the module. The overall project folder can act as a guide for generating new programs that utilize the SpacePoint Fusion. Note that SpacePoint Game Pad was developed on a Windows XP platform.

To view the source code in native format, Visual C++ Express can be downloaded at www.microsoft.com/express/vc/. Conversely, much of the source code is provided in hard-copy at the end of this document.

The key resources and source code in this project are:

- SpacePointGamePad.cpp
- UsbHidApi.h
- UsbHidApi.dll
- UsbHidApi.lib



Visual C++ Tree

SpacePointGamePad.cpp is the main source code to read sensor data from the SpacePoint Fusion USB interface. This data can be output to the display or logged to a text file. Hard-copy source code is provided at the end of this document.

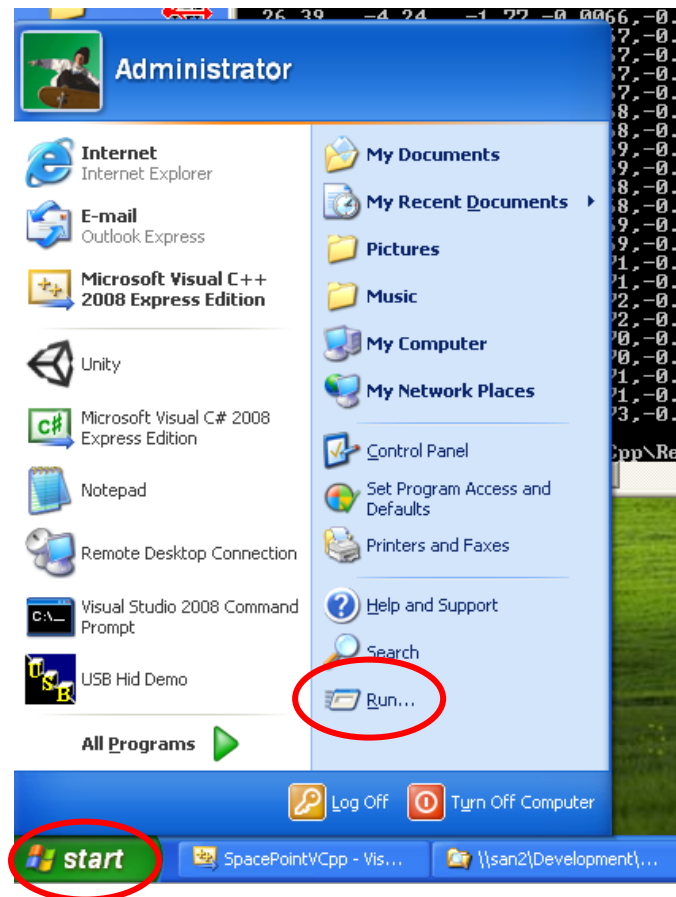
UsbHidApi.h includes the UsbHidApi library functions. Hard-copy source code is provided at the end of this document.

Running SpacePoint Game Pad

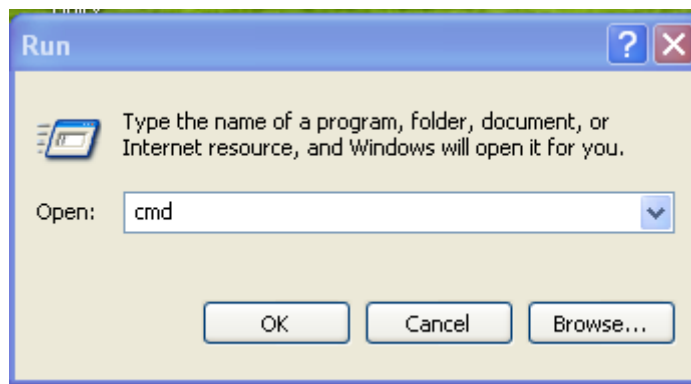
Instructions for running SpacePoint Game Pad follow. Note it is important to ensure the SpacePoint Fusion module is fully at rest for 3 seconds when plugging it into the computer's USB port, as the gyros initialize during this time.

1. Open a Command window by running cmd

Click on "Start", then "Run"



This will open the Run window. Type "cmd" then press <enter>.



This will open the Command window. If you are unfamiliar with commands available for the Command window, see <http://commandwindows.com/command3.htm> for an overview or <http://technet.microsoft.com/en-us/library/bb490890.aspx> for a more complete list.

2. In the Command window, change the directory to the “Release” folder where SpacepointGamePad.exe resides.

3. Run the executable file:

In the Command window (and in the “Release” directory) type the following:

```
SpacePointGamePad.exe [mode] [Nsample] [VID] [PID]
```

Where [mode] is either “0” for sensor data or “1” for calculated data, [Nsample] is the number of samples, [VID] is the vendor ID number, and [PID] is the product ID number. The VID and PID are 8447 and 256, and actually can be left blank. Example commands to obtain calculated data for 10 samples would be:

```
SpacePointGamePad.exe 1 10 8447 256
```

or

```
SpacePointGamePad.exe 1 10
```

This will display the results on the computer screen. To store the data to a log file use the pipe command “>>” as in the example below:

```
SpacePointGamePad.exe 1 100 >> test.log
```

This will create a file named test.log with calculated orientation data from 100 samples, and this file will be saved in the “Release” folder. The data will not be displayed as it is obtained. This data can be imported to a spreadsheet if desired.

Interpreting SpacePoint Game Pad Output

The displayed and logged outputs have the same format. The first line of output identifies the program, the second line is the header line for the data, and subsequent lines are the data. Example header and data lines are given below for the sensor data and for the calculated data, as well as explanations for the data.

Note that this section deals with data presented on the display or in the log file. The first section of this application note covers interpreting the USB/HID data streaming from the SpacePoint Fusion module, which is manipulated by the VC++ program to generate the data on the display or in the log file.

Sample sensor data is as follows:

<u>MagX</u>	<u>MagY</u>	<u>MagZ</u>	<u>AccX</u>	<u>AccY</u>	<u>AccZ</u>	<u>GyroR</u>	<u>GyroP</u>	<u>GyroY</u>	<u>Res1</u>	<u>Res2</u>	<u>LButton</u>	<u>RButton</u>
32785	33020	33445	32772	32766	33115	32240	32249	32242	1	211	0	0
32781	33017	33444	32772	32766	33114	32241	32250	32242	2	210	0	0
32779	33020	33445	32772	32766	33115	32240	32249	32243	3	208	0	0
32784	33016	33444	32772	32766	33115	32241	32250	32242	4	214	0	0
32780	33022	33445	32772	32766	33115	32240	32249	32243	5	222	0	0

All sensor outputs are 16-bit integers centered at 32768, and are representative of the raw sensor output. Res1 and Res2 are PNI reserved fields. LButton and RButton represent the status of the module's buttons: "1" is pressed, "0" is not pressed.

Sample calculated data is as follows:

<u>AX</u>	<u>AY</u>	<u>AZ</u>	<u>Q0</u>	<u>Q1</u>	<u>Q2</u>	<u>Q3</u>	<u>LButton</u>	<u>RButton</u>
0.9126	0.0837	0.1106	0.1061	-0.6336	0.0611	0.7639	1	0
0.9126	0.0837	0.1106	0.1061	-0.6336	0.0611	0.7639	1	0
0.9018	0.0782	0.1053	0.1059	-0.6336	0.0616	0.7639	1	0
0.9018	0.0782	0.1053	0.1059	-0.6336	0.0616	0.7639	1	0
0.9099	0.0837	0.1106	0.1055	,-0.6335	0.0621	0.7640	1	0

AX/Y/Z is the 3-axis normalized acceleration in g. Q0/1/2/3 are the normalized quaternion values. LButton and RButton represent the status of the module's buttons: "1" is pressed, "0" is not pressed.

Sample Code: SpacePointGamePad.cpp

```
// SpacePointGamePad.cpp : Defines the entry point for the console
application.
// This sample code is to demonstrate how to communicate with PNI's
SpacePoint.
// It is provided "as is" without express or implied warranty.
```

```
#include "stdafx.h"
#include <windows.h>
#include "UsbHidApi.h"
#include <math.h>
#include <iostream>
```

```
using namespace std;
```

```

void displayindata(unsigned char indata[21],int interfaceid,int Nbytes){
    int i, byteindex = 1;
    int rawaxes[11]; //raw has 9 axes, buttons, PNI byte
    float acc_scaled[3];
    float q_scaled[4];
    int buttons[2];

    if(interfaceid == 0){
        // Parse indata
        for (i = 0; i < 9; i++)
        {
            rawaxes[i] = (int)indata[byteindex] +
256*(int)indata[byteindex+1];
            byteindex += 2;
        }

        buttons[0] = indata[byteindex]&1;
        buttons[1] = (indata[byteindex]>>1)&1;
        rawaxes[9] = (indata[byteindex]>>4)&0xf;
        byteindex++;
        rawaxes[10] = indata[byteindex];

        for(i=0;i<9;i++)
            printf("%5i\t",rawaxes[i]);

        for(i=9;i<11;i++)
            printf("%3i\t",rawaxes[i]);

        printf("%1i\t%1i\n",buttons[0],buttons[1]);
    }

    if(interfaceid == 1){
        // Parse indata
        for (i = 0; i < 7; i++)
        {
            rawaxes[i] = (int)indata[byteindex] +
256*(int)indata[byteindex+1];
            byteindex += 2;
        }

        buttons[0] = indata[byteindex]&1;
        buttons[1] = (indata[byteindex]>>1)&1;
        rawaxes[7] = (indata[byteindex]>>4)&0xf;

        //16 bit raw values centered at 32768
        //acc_scaled = 6*(acc_received - 32768)/32768;
        for(i=0;i<3;i++)
            acc_scaled[i] = 6.0f* (rawaxes[i]-32768) / 32768.0f;

        printf("%7.4f,%7.4f,%7.4f,",acc_scaled[0],acc_scaled[1],acc_scaled[2]);

        //16 bit raw values centered at 32768
        //q_scaled = 3.0518e-005*(qraw - 32768)
        for(i=0;i<4;i++)
            q_scaled[i] = 3.0518e-005f*(rawaxes[3+i]-32768);
    }
}

```

```

    printf("%7.4f,%7.4f,%7.4f,%7.4f,",q_scaled[0],q_scaled[1],q_scaled[2],q
_scaled[3]);
    //printf("\n");
    printf("    %1i,\t%1i\n",buttons[0],buttons[1]);
    //printf(".");
}
}

int main(int argc, char* argv[])
{
    int connected = 0;
    int sample = 0;
    int mode = 0;
    int interfaceid = 0;
    int Nbytes = 0;
    int Nsamples = 10;
    int VID = 0x20ff;
    int PID = 0x0100;
    unsigned char indata[21];

    printf("PNI Corp, SpacePoint Game Pad data logger V1.00\n");
    if((argc != 3)&&(argc != 5)){
        cout << "usage: SpacePointGamePad Mode(0=raw,1=quaternion)
Nsamples <VID:8447=0x20ff> <PID:256=0x0100>" << endl;
    }
    else{

        interfaceid = atoi(argv[1]);
        Nsamples = atoi(argv[2]);
        if(argc>3){
            VID = atoi(argv[3]);
            PID = atoi(argv[4]);
        }
        //cout << VID << "\t" << PID << endl;
    }

    //printf("Opening SpacePoint, Interface %i\n",interfaceid);
    //Interface 0 = raw, Interface 1 = Kalman filter output
    if(interfaceid != 0)
        SetInterface(interfaceid);
    if (interfaceid == 0)
        printf
("MagX\tMagY\tMagZ\tAccX\tAccY\tAccZ\tGyroR\tGyroP\tGyroY\tRes1\tRes2\tLBut
ton\tRButton\n");
    if (interfaceid == 1)
        printf("    AX,    AY,    AZ,    Q0,    Q1,    Q2,    Q3,
LButton, RButton\n");

    if(connected = Open(VID,PID,NULL,NULL,NULL,1)) {
        //printf("Connected\n");
        sample = 0;

        while(connected) {

```

```

if(sample < Nsamples){
    if(Nbytes = Read(indata)>0){
        displayindata(indata,interfaceid,Nbytes);
        sample++;
    }
}
else{
    CloseRead();
    connected = 0;
}
}
}

return 0;
}

```

Sample Code: UsbHidApi.h

```

/*****
/*
/* File Name:   UsbHidApi.h
/*
/* Description:
/*
/*   This DLL provides a relatively simple method for accessing
/*   a USB HID device.  The device must specify HID reports for
/*   transferring data.
/*
/*   Client applications use the exported methods from the
/*   CUsbHidApi class to identify and access external
/*   HID device(s).  A new instance of the class must
/*   be created for each device being accessed.
/*
/*
/* Revision History:
/*
/* Name  Ver  Date      Description
/* ----  ---  ----      -
/* KAD   1.10  05/30/03  Initial version
/* KAD   1.12  06/19/03  Corrected first-chance exception
/* KAD   1.13  12/09/03  Corrected nag-screen bug
/* KAD   1.14  02/19/05  Modified calling convention to stdcall
/*                               instead of cdecl.
/*                               (Project -> Settings -> C/C++ tab ->
/*                               Category : Code Generation)
/* KAD   1.15  03/05/05  Corrected a problem with the module
/*                               definition file.
/* KAD   1.16  03/06/05  Added structure & wrapper functions for
/*                               Visual Basic access.  Also added library
/*                               deactivation after 30 days for shareware
/*                               version.
/* KAD   1.17  03/12/07  Added interface and collection capability.*
/* KAD   1.18  08/28/07  Added pre-compile switch for compatibility*
/*                               with ANSI C compilers like LabWindows/CVI.*
/****

```



```
/* KAD 1.18a 09/06/07 Removed ANSI C macro. Decided to use a */
/*                               separate header file for ANSI C use. */
/* KAD 1.18b 09/19/07 Changed period of registration reminder */
/*                               pop-ups from 10 minutes to 1 minute. */
/*****
//
#ifdef __USBHID_API_H__
#define __USBHID_API_H__
//
// Notes:
// -----
//
// This DLL provides the client application an easy method for accessing
// an HID device via USB link. The acts of reading and writing
// a USB device under Windows are significantly different and more
// involved than for other comm devices such as serial comm ports.
// For this reason, it seemed necessary to encapsulate the complexity
// of the interface within a DLL. This DLL provides all the required
// functions for accessing the device, while hiding the details
// of the implementation.
//
// The USB link is implemented as a Human Interface Device (HID) class
// function. As such, the DLL is dependent on the following Windows
// drivers:
//
//   hidclass.sys   hidparse.sys   hidusb.sys
//
// In addition to being USB-compliant, the host PC should have the
// latest, versions of these drivers. (Initial development was done
// using drivers from Windows 98, 2nd edition.)
//
// The module was built using Microsoft Visual C++, 6.0 as a Win32,
// non-MFC DLL. Since there is no dependence on the Microsoft
// Foundation Class (MFC), there is no need to copy additional MFC
// DLLs to your Windows system folder. Also, no dependence on MFC
// means the size of the DLL itself is minimal. (Although the DLL
// does not internally use the MFC, the client application is not
// restricted. Exported functions in the DLL may be called by either
// MFC or non-MFC applications.)
//
// The client application may link implicitly or explicitly with the
// DLL. In explicit linking, the client application makes function
// calls to explicitly load and unload the DLL's exported functions.
// In implicit linking, the application links to an import library
// (UsbHidApi.lib) and makes function calls just as if the functions
// were contained within the executable.
//
#define USBHIDAPI_DLL_NAME "UsbHidApi.dll"

// The following ifdef block is the standard way of creating macros which
// make exporting
// from a DLL simpler. All files within this DLL are compiled with the
USBHIDAPI_EXPORTS
// symbol defined on the command line. this symbol should not be defined
on any project
```



```
// that uses this DLL. This way any other project whose source files
include this file see
// USBHIDAPI_API functions as being imported from a DLL, whereas this DLL
sees symbols
// defined with this macro as being exported.
#ifdef USBHIDAPI_EXPORTS
#define USBHIDAPI_API __declspec(dllexport)
#else
#define USBHIDAPI_API __declspec(dllimport)
#endif

// This structure defines an entry in a device list
// The list describes the parameters associated with
// a device. It is mainly used with the GetList()
// function.
// modified 3/12/07
typedef struct {
    char DeviceName[50]; // Device name
    char Manufacturer[50]; // Manufacturer
    char SerialNumber[20]; // Serial number
    unsigned int VendorID; // Vendor ID
    unsigned int ProductID; // Product ID
    int InputReportLen; // Length of HID input report (bytes)
    int OutputReportLen; // Length of HID output report (bytes)
    int Interface; // Interface
    int Collection; // Collection
} mdeviceList;

// This structure was created to ease VB access. It alleviates
// some of the issues associated with inter-compiler data handling
// such as alignment.
// added 3/8/05
#pragma pack (push, 4)
typedef struct {
    char *DeviceName; // Device name
    char *Manufacturer; // Manufacturer
    char *SerialNumber; // Serial number
    unsigned int VendorID; // Vendor ID
    unsigned int ProductID; // Product ID
    int InputReportLen; // Length of HID input report (bytes)
    int OutputReportLen; // Length of HID output report (bytes)
    int Interface; // Interface
    int Collection; // Collection
} mdeviceList2;
#pragma pack (pop,4)

////////////////////////////////////
////////////////////////////////////
// These declarations define special non-member functions for VB
access.

////////////////////////////////////
////////////////////////////////////
```



```
extern "C" int _stdcall SetInstance(int instance);           // Set
object instance

extern "C" int _stdcall GetLibVersion(char *buf);           // Get DLL
version string

extern "C" void _stdcall ShowVersion(void);                // Show a
message box containing version

extern "C" int _stdcall Read(void *pBuf);                  // Read the
from the HID device.

extern "C" void _stdcall CloseRead(void);                  // Close
the read pipe

extern "C" void _stdcall CloseWrite(void);                 // Close
the write pipe

extern "C" int _stdcall Write(void *pBuf);                 // Write to
the HID device

extern "C" void _stdcall GetReportLengths (int *input_len, // Pointer
for storing input length                                     int *output_len); //
Pointer for storing output length

extern "C" void _stdcall SetCollection(int);                //
Specifies a collection (call prior to Open()) (0xffff = unspecified)

extern "C" int _stdcall GetCollection();                    //
Retrieves collection setting (0xffff = unspecified)

extern "C" void _stdcall SetInterface(int);                 //
Specifies an interface (call prior to Open()) (0xffff = unspecified)

extern "C" int _stdcall GetInterface();                     // Rerieves
interface setting (0xffff = unspecified)

extern "C" int _stdcall Open (unsigned int VendorID,        // Vendor
ID to search (0xffff if unused)                            unsigned int ProductID, //
Product ID to search (0xffff if unused)                    char *Manufacturer, //
Manufacturer (NULL if unused)                              char *SerialNum, //
Serial number to search (0xffff if unused)                 char *DeviceName, // Device
name to search (NULL if unused)                            int bAsync); // Set
TRUE for non-blocking read requests.

extern "C" int _stdcall GetList(unsigned int VendorID,      // Vendor
ID to search (0xffff if unused)
```

```

                unsigned int ProductID,        //
Product ID to search (0xffff if unused)
                char          *Manufacturer, //
Manufacturer          (NULL if unused)
                char          *SerialNum,     // Serial
number to search (NULL if unused)
                char          *DeviceName,   // Device
name to search      (NULL if unused)
                mdeviceList2 *pList,        //
Caller's array for storing matching device(s)
                int          nMaxDevices); // Size
of the caller's array list (no.entries)

////////////////////////////////////
////////////////////////////////////
// This class is exported from the UsbHidApi.dll
class USBHIDAPI_API CUsbHidApi {
public:

    // The serial number for the open device
    char  m_SerialNumber[20];

    // These variables define the required lengths for reading and writing
    // the device.
    unsigned short m_ReadSize;
    unsigned short m_WriteSize;

    // These variables define optional interface and/or collection values
search purposes
    // added 3/12/07
    int m_Interface;
    int m_Collection;

    // Constructor
    CUsbHidApi(void);

    // Destructor
    ~CUsbHidApi(void);

    // Get list of devices and their availability. Caller must supply a
    // pointer to a buffer that will hold the list of structure entries.
    // Must also supply an integer representing maximum no. of entries
    // his buffer can hold. Returns total number stored.
    int GetList(unsigned int VendorID,        // Vendor ID to search  (0xffff
if unused)
                unsigned int ProductID,     // Product ID to search (0xffff
if unused)
                char          *Manufacturer, // Manufacturer
(NULL if unused)
                char          *SerialNum,    // Serial number to search
(NULL if unused)
                char          *DeviceName,   // Device name to search
(NULL if unused)
                mdeviceList *pList,        // Caller's array for storing
matching device(s)

```

```

        int                nMaxDevices); // Size of the caller's array
list (no.entries)

// Opens a USB comm link to a HID device. Returns non-zero
// on success or 0 on failure. (Individual read and write handles
// are maintained internally.) If the caller desires to open
// a specific HID device, he must provide one or more
// specifiers (i.e., vendor ID, product ID, serial number,
// or device name. If a successful open occurs, the function
// returns TRUE. It returns FALSE otherwise.
int Open (unsigned int VendorID, // Vendor ID to search
(0xffff if unused)
        unsigned int ProductID, // Product ID to search
(0xffff if unused)
        char *Manufacturer, // Manufacturer (NULL
if unused)
        char *SerialNum, // Serial number to search
(0xffff if unused)
        char *DeviceName, // Device name to search (NULL
if unused)
        int bAsync); // Set TRUE for non-
blocking read requests.

// Sets an optional device interface ID (e.g., 0) for search purposes.
// This method is used to pin-point a device to be opened.
// Use this method when a USB device has multiple interfaces.
// Must be called prior to calling the Open() method.
// added 3/12/07
void SetInterface (int iface); // Interface (-1 if unused)

// Returns the device interface ID that was set using SetInterface().
// added 3/12/07
int GetInterface (void);

// Sets an optional collection ID (e.g., 0) for search purposes.
// This method is used to pin-point a device to be opened.
// Use this method when a USB device has multiple collections.
// Must be called prior to calling the Open() method.
// added 3/12/07
void SetCollection (int col); // Collection (-1 if unused)

// Returns the collection ID that was set using SetCollection().
// added 3/12/07
int GetCollection (void);

// Close the read pipe
void CloseRead(void);

// Close the write pipe
void CloseWrite(void);

// Read the from the HID device. The number of bytes read is
// determined by the input report length specified by the device.
// Depending on how the device was opened, the call may perform
// blocking or non-blocking reads. Refer to Open() for details.
// On success, the call returns the number of bytes actually read.

```

```

// A negative return value indicates an error (most likely means
// the USB cable has been disconnected or device was powered off).
// (Note: The first byte location is usually a report ID
// [typically = 0]. The caller must account for this value in
// the read buffer.)
int Read(void *pBuf); // Buffer for storing bytes

// Write a report to the HID device. The number of bytes to
// write depend on the output report length specified by the
// device. Returns number of bytes actually written. A negative
// return value indicates an error (most likely means the
// USB cable has been disconnected or device was powered off).
// (Note: The first byte location is usually a report ID
// [typically = 0]. The caller must ensure this value is
// prepended to the buffer.)
int Write(void *pBuf); // Buffer containing bytes to
write

// This function retrieves the lengths of input- and output-reports
// for the device. These values determine the I/O lengths for the
// Read() and Write() functions. The device must be opened prior
// to calling this routine. Alternatively, you can just use m_ReadSize
// and m_WriteSize.
void GetReportLengths (int *input_len, // Pointer for storing input
length
int *output_len); // Pointer for storing output
length

// This function retrieves the current library version and populates a
string
int GetLibVersion(LPSTR buf);

// This function displays the current library version in a standard
message box
void ShowVersion(void);

// Private (internal) declarations
private:

// This function reads an input report from an open device.
// The call will block until any number of bytes is retrieved,
// up to the maximum of nBytesToRead. On successful completion,
// the function returns an integer representing the
// number of bytes read (usually the input report length
// defined by the device). (Note: The read buffer must
// be large enough to accommodate the report length. This
// number is located in m_ReadSize.)
int ReadSync(void *pBuf); // Buffer for storing bytes

// Read input report from an open device. If no data
// is currently available, the call will not block,
// but will return 0. On successful completion,
// the function returns an integer representing the
// number of bytes read (usually the input report length
// defined by the device). The function returns -1 if

```



```
// disconnect is detected. (Note: The read buffer must
// be large enough to accommodate the report length. This
// number is located in m_ReadSize.)
int ReadAsync(void *pBuf);          // Buffer for storing bytes

// Read/write handles. While a single handle could suffice for
// both operations, 2 handles have been created to allow the client
// application to utilize separate threads for reading and writing.
HANDLE ReadHandle;
HANDLE WriteHandle;
};

#endif // #ifndef __USBHID_API_H__
```