C# vJoy Feeder SDK

Version 2.0.5 – March 2015

Table of Contents

This SDK includes all that is needed to write a feeder for vJoy version 2.0.1
Check for the latest SDK.

File listing:

c#	C# SDK (This folder)
x86	Library folder (x86)
x86\vJoyInterface.dll	vJoy Interface DLL file (32-bit version)
x86\vJoyInterface.pdb	Program Database – Use it for debugging (32-bit version)
x86\vJoyInterfaceWrap.dll	vJoy c# wrapper DLL
x64	Library folder (x64)
x64\vJoyInterface.dll	vJoy Interface DLL file (64-bit version)
x64\vJoyInterface.pdb	Program Database – Use it for debugging (64-bit version)
x64\vJoyInterfaceWrap.dll	vJoy c# wrapper DLL
FeederDemoCS	Demo Feeder Project (Visual Studio 2008 Express)
FeederDemoCS\Program.cs	C# code that demonstrates writing a simple feeder
FeederDemoCS\FeederDemoCS.csproj	Demo Feeder Project file (Visual Studio 2008 Express)
FeederDemoCS\FeederDemoCS.sln	Demo Feeder solution file (Visual Studio 2008 Express)
FeederDemoCS\Properties\AssemblyInfo.cs	Demo Feeder properties file (Visual Studio 2008 Express)

Fundamentals:

This interface and example will enable you to write a C# vJoy feeder.
To write a C/C++ refer to ReadMe file in parent folder.
Features introduced in version 2.0.5 are marked with [2.0.5]

It is advisable to start your feeder from the supplied example and make the needed changes. Here are the five basic steps you might want to follow:

Test Driver:	Check that the driver is installed and enabled. Obtain information about the driver. An installed driver implies at least one vJoy device. [2.0.5] Test if driver matches interface DLL file (vJoyInterface.dll)
Test Virtual Device(s):	Get information regarding one or more devices. Read information about a specific device capabilities: Axes, buttons and POV hat switches.
Device acquisition:	Obtain status of a vJoy device. Acquire the device if the device status is *owned* or is *free.*
Updating:	Inject position data to a device (as long as the device is owened by the feeder). Position data includes the position of the axes, state of the buttons and state of the POV hat switches.
Relinquishing the device:	The device is *owned* by the feeder and cannot be fed by another application until relinquished.

Notes:

1. The interface library file (vJoyInterface.dll) and the wrapper library (vJoyInterfaceWrap.dll) must be placed together.
2. The feeder must use **using** directive:
   ```
   using vJoyInterfaceWrap;
   ```
3. Wrapper only class is vJoy: Start your application by creating a **vJoy** object:
   ```
   joystick = new vJoy();
   ```

Reccomended Practices:

Test vJoy Driver:

Before you start, check if the vJoy driver is installed and check that it is what you expected:

```
joystick = new vJoy();

// Get the driver attributes (Vendor ID, Product ID, Version Number)
if (!joystick.vJoyEnabled())
  {
   Console.WriteLine("vJoy driver not enabled: Failed Getting vJoy attributes.\n");
   return;
  }
  else
   Console.WriteLine("Vendor: {0}\nProduct :{1}\nVersion Number:{2}\n",
   joystick.GetvJoyManufacturerString(),
   joystick.GetvJoyProductString(),
   joystick.GetvJoySerialNumberString());
```

Test Interface DLL matches vJoy Driver:

[2.0.5]
Before you start, check if file vJoyInterface.dll that you link to matches the vJoy driver that is installed. It is recommended that their version numbers will be identical.

```
// Test if DLL matches the driver
UInt32 DllVer = 0, DrvVer = 0;
bool match = joystick.DriverMatch(ref DllVer, ref DrvVer);
if (match)
      Console.WriteLine("Version of Driver Matches DLL Version ({0:X})\n", DllVer);
else
      Console.WriteLine("Version of Driver ({0:X}) does NOT match DLL Version ({1:X})\n",
DrvVer, DllVer);
```

If you are not interested in the actual values of the respective version numbers, you can simplify your code by passing NULL to both function parameters.

Test vJoy Virtual Devices:

Check if device is installed and what its state:

```
// Get the state of the requested device
VjdStat status = joystick.GetVJDStatus(id);

switch (status)
{
      case VjdStat.VJD_STAT_OWN:
            Console.WriteLine("vJoy Device {0} is already owned by this feeder\n", id);
            break;
      case VjdStat.VJD_STAT_FREE:
            Console.WriteLine("vJoy Device {0} is free\n", id);
            break;
      case VjdStat.VJD_STAT_BUSY:
            Console.WriteLine(
               "vJoy Device {0} is already owned by another feeder\nCannot continue\n", id);
            return;
      case VjdStat.VJD_STAT_MISS:
            Console.WriteLine(
                "vJoy Device {0} is not installed or disabled\nCannot continue\n", id);
            return;
      default:
            Console.WriteLine("vJoy Device {0} general error\nCannot continue\n", id);
            return;
};
```

Now make sure that the axes, buttons (and POV hat switches) are as expected:

```
///// vJoy Device properties
int nBtn = joystick.GetVJDButtonNumber(id);
int nDPov = joystick.GetVJDDiscPovNumber(id);
int nCPov = joystick.GetVJDContPovNumber(id);
bool X_Exist = joystick.GetVJDAxisExist(id, HID_USAGES.HID_USAGE_X);
bool Y_Exist = joystick.GetVJDAxisExist(id, HID_USAGES.HID_USAGE_Y);
bool Z_Exist = joystick.GetVJDAxisExist(id, HID_USAGES.HID_USAGE_Z);
bool RX_Exist = joystick.GetVJDAxisExist(id, HID_USAGES.HID_USAGE_RX);

prt = String.Format("Device[{0}]: Buttons={1}; DiscPOVs:{2}; ContPOVs:{3}",\
           id, nBtn, nDPov, nCPov);
Console.WriteLine(prt);
```

Acquire the vJoy Device:

Until now you just made inquiries about the system and about the vJoy device status. In order to change the position of the vJoy device you need to Acquire it (if it is not already owned):

```
///// Write access to vJoy Device - Basic
VjdStat status;
status = joystick.GetVJDStatus(id);
// Acquire the target
if ((status == VjdStat.VJD_STAT_OWN) ||
            ((status == VjdStat.VJD_STAT_FREE) && (!  joystick.AcquireVJD(id))))
      prt = String.Format("Failed to acquire vJoy device number {0}.", id);
else
      prt = String.Format("Acquired: vJoy device number {0}.", id);
Console.WriteLine(prt);
```

Feed vJoy Device:

The time has come to do some real work: feed the vJoy device with position data.

There are two approaches:

1. **Efficient**: Collect position data, place the data in a position structure then finally send the data to the device.
2. **Robust**: Reset the device once then send the position data for every control (axis, button,POV) at a time.

The first approach is more efficient but requires more code to deal with the position structure that may change in the future.

The second approach hides the details of the data fed to the device at the expence of exessive calls to the device driver.

Efficient:

```
while (true)
{
          // Feed the device id
   iReport.bDevice = (byte)id;
          // Feed position data per axis
   iReport.AxisX = X;
   iReport.AxisY = Y;
   iReport.AxisZ = Z;
   iReport.AxisZRot = ZR;
   iReport.AxisXRot = XR;
   // Set buttons one by one
   iReport.Buttons = (uint)(0x1 <<  (int)(count / 20));

         if (ContPovNumber>0)
         {
               // Make Continuous POV Hat spin
               iReport.bHats           = (count*70);
               iReport.bHatsEx1  = (count*70)+3000;
               iReport.bHatsEx2  = (count*70)+5000;
               iReport.bHatsEx3  = 15000 - (count*70);
               if ((count*70) > 36000)
               {
                     iReport.bHats =    0xFFFFFFFF; // Neutral state
               iReport.bHatsEx1 = 0xFFFFFFFF; // Neutral state
               iReport.bHatsEx2 = 0xFFFFFFFF; // Neutral state
               iReport.bHatsEx3 = 0xFFFFFFFF; // Neutral state
               };
         }
         else
         {
               // Make 5-position POV Hat spin
               pov[0] = (byte)(((count / 20) + 0) %4);
               pov[1] = (byte)(((count / 20) + 1) % 4);
         pov[2] = (byte)(((count / 20) + 2) % 4);
         pov[3] = (byte)(((count / 20) + 3) % 4);

               iReport.bHats =
                  (uint)(pov[3]<<12)|(uint)(pov[2]<<8)|(uint)(pov[1]<<4)|(uint)pov[0];
               if ((count) > 550)
                     iReport.bHats = 0xFFFFFFFF; // Neutral state
         };

 /*** Feed the driver with the position packet  ***/
 joystick.UpdateVJD(id, ref iReport)
 System.Threading.Thread.Sleep(20);
 count++;
 if (count > 640) count = 0;

 X += 150; if (X > maxval) X = 0;
 Y += 250; if (Y > maxval) Y = 0;
 Z += 350; if (Z > maxval) Z = 0;
 XR += 220; if (XR > maxval) XR = 0;
 ZR += 200; if (ZR > maxval) ZR = 0;

}; // While
```

if the structure changes in the future then the code will have to change too.

Robust:

```
joystick.ResetVJD(id);// Reset this device to default values
while (true)// Feed the device in endless loop
{
  // Set position of 4 axes
  res = joystick.SetAxis(X, id, HID_USAGES.HID_USAGE_X);
  res = joystick.SetAxis(Y, id, HID_USAGES.HID_USAGE_Y);
  res = joystick.SetAxis(Z, id, HID_USAGES.HID_USAGE_Z);
  res = joystick.SetAxis(XR, id, HID_USAGES.HID_USAGE_RX);
  res = joystick.SetAxis(ZR, id, HID_USAGES.HID_USAGE_RZ);

  // Press/Release Buttons
  res = joystick.SetBtn(true, id, count / 50);
  res = joystick.SetBtn(false, id, 1 + count / 50);

  // If Continuous POV hat switches installed - make them go round
  // For high values - put the switches in neutral state
  if (ContPovNumber>0)
  {
    if ((count * 70) < 30000)
    {
      res = joystick.SetContPov(((int)count * 70), id, 1);
      res = joystick.SetContPov(((int)count * 70) + 2000, id, 2);
      res = joystick.SetContPov(((int)count * 70) + 4000, id, 3);
      res = joystick.SetContPov(((int)count * 70) + 6000, id, 4);
    }
    else
    {
      res = joystick.SetContPov(-1, id, 1);
      res = joystick.SetContPov(-1, id, 2);
      res = joystick.SetContPov(-1, id, 3);
      res = joystick.SetContPov(-1, id, 4);
    };
  };
  // If Discrete POV hat switches installed - make them go round
  // From time to time - put the switches in neutral state
  if (DiscPovNumber>0)
  {
    if (count < 550)
    {
      joystick.SetDiscPov((((int)count / 20) + 0) % 4, id, 1);
      joystick.SetDiscPov((((int)count / 20) + 1) % 4, id, 2);
      joystick.SetDiscPov((((int)count / 20) + 2) % 4, id, 3);
      joystick.SetDiscPov((((int)count / 20) + 3) % 4, id, 4);
    }
    else
    {
      joystick.SetDiscPov(-1, id, 1);
      joystick.SetDiscPov(-1, id, 2);
      joystick.SetDiscPov(-1, id, 3);
      joystick.SetDiscPov(-1, id, 4);
    };
  };
  System.Threading.Thread.Sleep(20);
} // While (Robust)
```

This code is readable and does not relay on any specific structure. However, the driver is updated with every *SetAxis()* and every *SetBtn()*.

Relinquish the vJoy Device:

You must relinquish the device when the driver exits:

```
joystick.RelinquishVJD(iInterface);
```

Detecting Changes

[2.0.5]

It is sometimes necessary to detect changes in the number of available vJoy devices.

You may define a callback function that will be called whenever such a change occurs. In order for it to be called, the user-defined callback function should first be registered by calling function *RegisterRemovalCB* as in the following example:

```
joystick.RegisterRemovalCB(ChangedCB, label2);
```

Where ChangedCB is the user-defined callback function and label2 is some C# object.

An example to an implementation of the user-defined callback function ChangedCB:

```
void CALLBACK ChangedCB(bool Removed, bool First, object userData)
{
            Label l = userData as Label;
            int id = 1;
            int nBtn = joystick.GetVJDButtonNumber(id);



            // Final values after the last arival
            if (!removal && !first)
            {
                prt = String.Format("Device[{0}]: Buttons={1}" id, nBtn);
                l.Text = prt;
            }

            // Temporary message during intermediate states
            else
            {
                prt = String.Format("Device[{0}]: Wait ...", id);
                l.Text = prt;
            }

}
```

This function is called when a process of vJoy device removal starts or ends and when a process of vJoy device arrival starts or ends. The function must return as soon as possible.

- When a process of vJoy device removal starts, Parameter *Removed*=TRUE and parameter *First*=TRUE.
- When a process of vJoy device removal ends, Parameter *Removed*=TRUE and parameter *First*=FALSE.
- When a process of vJoy device arrival starts, Parameter *Removed*=FALSE and parameter *First*=TRUE.
- When a process of vJoy device arrival ends, Parameter *Removed*= FALSE and parameter *First*=FALSE .

Parameter *userData* is always an object registered as second parameter of function *RegisterRemovalCB*.

Interface Function Reference:

General driver data

The following functions return general data regarding the installed vJoy device driver. It is recommended to call them when starting your feeder.

```
bool vJoyEnabled();
```
Returns true if vJoy version 2.x is installed and enabled.

```
short GetvJoyVersion();
```
Return the version number of the installed vJoy. To be used only after `vJoyEnabled()`

```
string    GetvJoyProductString();
string    GetvJoyManufacturerString();
string    GetvJoySerialNumberString();
```
To be used only after `vJoyEnabled()`

[2.0.5]
```
bool      DriverMatch(ref UInt32 DllVer, ref UInt32 DrvVer);
```
Returns TRUE if vJoyInterface.dll file version and vJoy Driver version are identical. Otherwise returns FALSE.

Optional output parameter *DllVer*: If a reference to 32-bit unsigned integer is passed then the value of the **DLL file** version will be written to this parameter (e.g. 0x205).
Optional output parameter *DrvVer*: If a reference to 32-bit unsigned integer is passed then the value of the **Driver** version will be written to this parameter (e.g. 0x205).

[2.0.5]
```
void RegisterRemovalCB(RemovalCbFunc cb, object data);
```
This function registers a user-defined **cb** callback fuction that is called everytime a vJoy device is added or removed.
Paremeter *cb* is a reference to the user-defined callback function.
Parameter *data* is a pointer to a user-defined object. The callback function recieves this object as its third parameter.
The user-defined callback function type definition:
```
void RemovalCbFunc(bool complete, bool First, object userData);
```
More in section Detecting Changes.

Write access to vJoy Device

The following functions access the virtual device by its ID (rID). The value of rID may vary between 1 and 16.
There may be more than one virtual device installed on a given system.
VJD stands for *Virtual Joystick Device*.

```
VjdStat   GetVJDStatus(UInt32 rID);
```
Returns the status of the specified device
The status can be one of the following values:

- VJD_STAT_OWN // The vJoy Device is owned by this application.
- VJD_STAT_FREE // The vJoy Device is NOT owned by any application (including this one).
- VJD_STAT_BUSY // The vJoy Device is owned by another application.
  // It cannot be acquired by this application.
- VJD_STAT_MISS // The vJoy Device is missing. It either does not exist or the driver is disabled.
- VJD_STAT_UNKN // Unknown

```
bool AcquireVJD(UInt32 rID);
```
Acquire the specified device.
Only a device in state VJD_STAT_FREE can be acquired.
If acquisition is successful the function returns TRUE and the device status becomes VJD_STAT_OWN.

```
void RelinquishVJD(UInt32 rID);
```
Relinquish the previously acquired specified device.
Use only when device is state VJD_STAT_OWN.
State becomes VJD_STAT_FREE immediately after this function returns.

```
bool UpdateVJD(UInt32 rID, ref JoystickState pData);
```
Update the position data of the specified device.
Use only after device has been succesfully acquired.
Input parameter is a reference to structure of type JoystickState that holds the position data.
Returns **true** if device updated.

vJoy Device properties

The following functions receive the virtual device ID (rID) and return the relevant data.
The value of rID may vary between 1 and 16. There may be more than one virtual device installed on a given system.
The return values are meaningful only if the specified device exists
VJD stands for Virtual Joystick Device.

```
int  GetVJDButtonNumber(uint rID);
```
Returns the number of buttons in the specified device.
If function succeeds, returns the number of buttons in the specified device. Valid values are 0 to 128
[2.0.5]If function fails, returns a negative error code:
- NO_HANDLE_BY_INDEX
- BAD_PREPARSED_DATA
- NO_CAPS
- BAD_N_BTN_CAPS
- BAD_BTN_CAPS
- BAD_BTN_RANGE

```
int  GetVJDDiscPovNumber(uint rID);
```
Returns the number of discrete-type POV hats in the specified device
Discrete-type POV Hat values may be North, East, South, West or neutral
Valid values are 0 to 4 (in version 2.0.1)

```
int  GetVJDContPovNumber(uint rID);
```
Returns the number of continuous-type POV hats in the specified device
continuous-type POV Hat values may be 0 to 35900
Valid values are 0 to 4 (in version 2.0.1)

```
bool GetVJDAxisExist(UInt32 rID, HID_USAGES Axis);
```
Returns TRUE is the specified axis exists in the specified device
Axis values can be:
```
HID_USAGES.HID_USAGE_X          // X Axis
HID_USAGES.HID_USAGE_Y          // Y Axis
HID_USAGES.HID_USAGE_Z          // Z Axis
HID_USAGES.HID_USAGE_RX         // Rx Axis
HID_USAGES.HID_USAGE_RY         // Ry Axis
```

```
HID_USAGES.HID_USAGE_RZ              // Rz Axis
HID_USAGES.HID_USAGE_SL0             // Slider 0
HID_USAGES.HID_USAGE_SL1             // Slider 1
HID_USAGES.HID_USAGE_WHL // Wheel
```

Robust write access to vJoy Devices

The following functions receive the virtual device ID (rID) and return the relevant data.
These functions hide the details of the position data structure by allowing you to alter the value of a specific control.
The downside of these functions is that you inject the data to the device serially as opposed to function *UpdateVJD()*.
The value of rID may vary between 1 and 16. There may be more than one virtual device installed on a given system.

```
bool ResetVJD(UInt32 rID);
```
Resets all the controls of the specified device to a set of values.
These values are hard coded in the interface DLL and are currently set as follows:

- Axes X,Y & Z: Middle point.
- All other axes: 0.
- POV Switches: Neutral (-1).
- Buttons: Not Pressed (0).

```
bool  ResetAll();
```
Resets all the controls of the all devices to a set of values.
See function Reset VJD for details.

```
bool ResetButtons(UInt32 rID);
```
Resets all buttons (To 0) in the specified device.

```
bool ResetPovs(UInt32 rID);
```
Resets all POV Switches (To -1) in the specified device.

```
bool SetAxis(Int32 Value, UInt32 rID, HID_USAGES Axis);
```
Write Value to a given axis defined in the specified VDJ.
Axis values can be:

```
 HID_USAGES.HID_USAGE_X          // X Axis
HID_USAGES.HID_USAGE_Y           // Y Axis
HID_USAGES.HID_USAGE_Z           // Z Axis
HID_USAGES.HID_USAGE_RX          // Rx Axis
HID_USAGES.HID_USAGE_RY          // Ry Axis
HID_USAGES.HID_USAGE_RZ          // Rz Axis
HID_USAGES.HID_USAGE_SL0         // Slider 0
HID_USAGES.HID_USAGE_SL1         // Slider 1
HID_USAGES.HID_USAGE_WHL // Wheel
```

```
bool SetBtn(bool Value, UInt32 rID, uint nBtn);
```
Write Value (true or false) to a given button defined in the specified VDJ.
nBtn can range 1-32

```
bool SetDiscPov(Int32 Value, UInt32 rID, uint nPov);
```
Write Value to a given discrete POV defined in the specified VDJ
Value can be one of the following:
0: North (or Forwards)
1: East (or Right)
2: South (or backwards)
3: West (or left)
-1: Neutral (Nothing pressed)
nPov selects the destination POV Switch. It can be 1 to 4

```
bool SetContPov(Int32 Value, UInt32 rID, uint nPov);
```
Write Value to a given continuous POV defined in the specified VDJ
Value can be in the range: -1 to 35999. It is measured in units of one-hundredth a degree. -1 means Neutral (Nothing pressed).
nPov selects the destination POV Switch. It can be 1 to 4

Build & Deploy:

The quickest way to build your project is to start from the supplied demo project written in C# under Visual Studio 2008 Express. It will compile as-is for x86/x64 target machines.

When you deploy your feeder, don't forget to supply the user with files vJoyInterface.dll and vJoyInterfaceWrap.dll of the correct bitness. They should be located on the target machine's DLL search path. Usually meaning the same directory as your feeder.

Location of vJoyInterface.dll

[2.0.5]
vJoy folders are pointed at by registry Entries located under key:
`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{8E31F76F-74C3-47F1-9550-E041EEDC5FBB}_is1`

Entry	Default Value	Notes
InstallLocation	C:\Program Files\vJoy\	vJoy root folder: Location of vJoy driver installer and uninstaller
DllX64Location	C:\Program Files\vJoy\x64	• Location of 64-bit utilities and libraries • Only on 64-bit Machines
DllX86Location	C:\Program Files\vJoy\x86	• Location of 32-bit utilities and libraries • On 32-bit and 64-bit Machines

Note that on 64-bit machine you are capable of developing both 32-bit and 64-bit feeders.

You can assume that DLL files are located in sub-folders x64 and x32 under vJoy root folder.

Logging [2.0.5]

Logging of vJoyInterface.dll activity into a log file is an option.
Use this feature for debugging purposes only. It accumulates data into the log file and generally slows down the system.

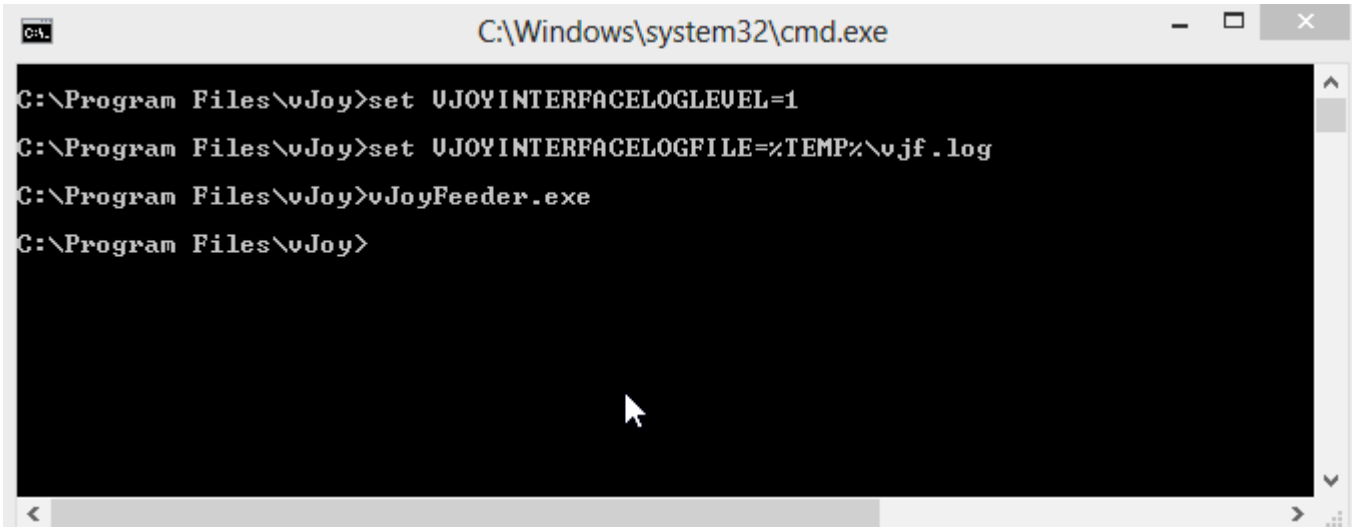
This feature is intended both for helping you develop your feeder and to collect data at the user's location – provided the user is willing to trigger logging for you. By default, logging state is OFF.

Start/Stop Logging.

To start logging, there are one or two system environment variables that have to be changed before the feeder (Or any other application calling vJoyInterface.dll) is started.

- VJOYINTERFACELOGLEVEL:
  Any positive value will trigger logging.
  Set to 0 to stop logging.
- VJOYINTERFACELOGFILE (Optional):
  If set, this is the full path to the log file.
  Default Path: %TEMP%\vJoyInterface.log

Example:



Notes:

- This session of vJoyFeeder will log into the given file.
- If the file exists, it will append the new data to the existing file.
- To stop logging, kill vJoyFeeder and then close this window.

Limitations:

- Logging begins on the application's first call to function AcquireVJD()
- If VJOYINTERFACELOGFILE is not defined, all applications that call AcquireVJD() will write to the same default output file.

Log File

The log file contains information about vJoyInterface.dll values, states and functions. It is mainly useful in conjunction with the code.

Here is a snippet of a log file:

```
[04988]Info: GetHandleByIndex(index=3) - Starting

[04988]Info: GetHandleByIndex(index=3) - Exit OK (Handle to \\?\hid#hidclass&col01#1&2d595ca7&db&0000#{4d1e55b2-
f16f-11cf-88cb-001111000030})




[03088]Process:"D:\WinDDK\vJoy-2.0.5\apps\vJoyFeeder\x64\Release\vJoyFeeder.exe"



[03088]Info: OpenDeviceInterface(9) - DevicePath[0]=\\?\{d6e55ca0-1a2e-4234-aaf3-
3852170b492f}#vjoyrawpdo#1&2d595ca7&db&vjoyinstance00#{781ef630-72b2-11d2-b852-00c04fad5101}\device_001

[03088]Info: isRawDevice(9) - Compare \\?\{d6e55ca0-1a2e-4234-aaf3-
3852170b492f}#vjoyrawpdo#1&2d595ca7&db&vjoyinstance00#{781ef630-72b2-11d2-b852-00c04fad5101}\device_001 with 001(d=1)

[03088]Info: OpenDeviceInterface(9) - DevicePath[1]=\\?\{d6e55ca0-1a2e-4234-aaf3-
3852170b492f}#vjoyrawpdo#1&2d595ca7&db&vjoyinstance00#{781ef630-72b2-11d2-b852-00c04fad5101}\device_002

[03088]Info: isRawDevice(9) - Compare \\?\{d6e55ca0-1a2e-4234-aaf3-
3852170b492f}#vjoyrawpdo#1&2d595ca7&db&vjoyinstance00#{781ef630-72b2-11d2-b852-00c04fad5101}\device_002 with 002(d=2)

[03088]Info: OpenDeviceInterface(9) - DevicePath[2]=\\?\{d6e55ca0-1a2e-4234-aaf3-
3852170b492f}#vjoyrawpdo#1&2d595ca7&db&vjoyinstance00#{781ef630-72b2-11d2-b852-00c04fad5101}\device_003

[03088]Info: isRawDevice(9) - Compare \\?\{d6e55ca0-1a2e-4234-aaf3-
3852170b492f}#vjoyrawpdo#1&2d595ca7&db&vjoyinstance00#{781ef630-72b2-11d2-b852-00c04fad5101}\device_003 with 003(d=3)
```

You can see the end of one process (Process ids are in brackets) and the beginning of a second process. The first line referring the second project is highlighted, and it indicates the command this process is carrying out.

Every line in the log file starts with the process id and followed by an error level string such as Info and a column.

The next string is usually the name of the function (e.g. `isRawDevice`) and its significant parameters.

For full understanding of the printout you should refer to the source file.