## vJoy Feeder/Receptor SDK

*Version 2.1.8 Release – November 2016*

# Table of Contents



This SDK includes all that is needed to write a feeder for vJoy version 2.1.8
Check for the latest SDK.

### *Files listing:*

| | |
|---|---|
| **inc** | Include folder |
| **inc\public.h** | vJoy general public definitions |
| **inc\vjoyinterface.h** | Interface function declaration for vJoyInterface.dll |
| | |
| **lib** | Library folder (x86) folder |
| **lib\vJoyInterface.dll** | vJoy Interface module – must be included with the feeder (x86) |
| **lib\vJoyInterface.lib** | Import library – you must link your feeder to it (x86) |
| **lib\vJoyInterface.pdb** | Program Database – Use it for debugging (x86) |
| | |
| **lib\amd64** | Library folder (x64) folder |
| **lib\amd64\vJoyInterface.dll** | vJoy Interface module – must be included with the feeder (x64) |

| | |
|---|---|
| **lib\amd64\vJoyInterface.lib** | Import library – you must link your feeder to it (x64) |
| **lib\amd64\vJoyInterface.pdb** | Program Database – Use it for debugging (x64) |
| | |
| **src** | Sources of an example feeder folder |
| **src\vJoyClient.cpp** | Sources |
| **src\vJoyClient.sln** | VS2008 Express solution |
| **src\vJoyClient.vcproj** | VS2008 Express project |
| **src\stdafx.h** | Additional header files |
| | |
| **c#** | C# SDK folder |
| **x86** | Library folder (x86) folder |
| **x64** | Library folder (x64) folder |
| **WrapperTest** | Demo Wrapper Project (Visual Studio 2008 Express) folder |
| **ReadMe.pdf** | C# SDK Read Me file |

## *Fundamentals:*

This interface and example will enable you to write a C/C++ vJoy feeder/receptor.
Features introduced in version 2.1.6 to 2.1.8 are marked with [New]
To write a C# refer to manual in C# folder.

## *Feeder:*

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.<br>Obtain information about the driver.<br>An installed driver implies at least one vJoy device.<br>Test if driver matches interface DLL file |
| **Test Virtual Device(s)**: | Get information regarding one or more devices.<br>Read information about a specific device capabilities: Axes, buttons and POV hat switches. |
| **Device acquisition**: | Obtain status of a vJoy device.<br>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 owned by the feeder).<br>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. |

```
WORD VerDll, VerDrv;
if (!DriverMatch(&VerDll, &VerDrv))
	_tprintf("Failed\r\nvJoy Driver (version %04x) does not match vJoyInterface DLL
(version %04x)\n", VerDrv ,VerDll);
else
	_tprintf( "OK - vJoy Driver and vJoyInterface DLL match vJoyInterface DLL
(version %04x)\n", VerDrv);
	{
		_tprintf("Vendor: %S\nProduct :%S\nVersion Number:%S\n",\
		TEXT(GetvJoyManufacturerString()),\
		TEXT(GetvJoyProductString()),\
		TEXT(GetvJoySerialNumberString()));
	};
```

## *Recommended Practices:*

## *Test vJoy Driver:*

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

## *Test Interface DLL matches vJoy Driver:*

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.
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.

```
	// Get the state of the requested device
	VjdStat status = GetVJDStatus(iInterface);
	switch (status)
	{
	case VJD_STAT_OWN:
		_tprintf("vJoy Device %d is already owned by this feeder\n", iInterface);
		break;
	case VJD_STAT_FREE:
		_tprintf("vJoy Device %d is free\n", iInterface);
		break;
	case VJD_STAT_BUSY:
		_tprintf("vJoy Device %d is already owned by another feeder\nCannot continue\n",
iInterface);
		return -3;
	case VJD_STAT_MISS:
		_tprintf("vJoy Device %d is not installed or disabled\nCannot continue\n",
iInterface);
		return -4;
	default:
		_tprintf("vJoy Device %d general error\nCannot continue\n", iInterface);
		return -1;
	};
```

### *Test vJoy Virtual Devices:*

Check which devices are installed and what their state is it:
Now make sure that the axes, buttons (and POV hat switches) are as expected:

```
	// Acquire the target
	if ((status == VJD_STAT_OWN) || ((status == VJD_STAT_FREE) &&
(!AcquireVJD(iInterface))))
	{
		_tprintf("Failed to acquire vJoy device number %d.\n", iInterface);
		return -1;
	}
	else
	{
		_tprintf("Acquired: vJoy device number %d.\n", iInterface);
	}
```

### *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):

```
	/*** Create the data packet that holds the entire position info ***/

	// Set the device ID
	id = (BYTE)iInterface;
	iReport.bDevice = id;

	// Set values in four axes (Leave the rest in default state)
	iReport.wAxisX=X;
	iReport.wAxisY=Y;
	iReport.wAxisZ=Z;
	iReport.wAxisZRot=ZR;

	// Set buttons one by one
	iReport.lButtons = 1<<count/20;

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

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

## *Feed vJoy Device:*

1. The time has come to do some real work: feed the vJoy device with position data.
2. 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. The second approach hides the details of the data fed to the device at the expense of excessive calls to the device driver.

**Efficient**:

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

```
void CALLBACK ChangedCB(BOOL Removed, BOOL First, PVOID data)
{
	HWND hDlg = (HWND)data;
	PostMessage(hDlg, WM_VJOYCHANGED, (WPARAM)Removed, (LPARAM)First);
}
	{
		for(int i=0;i<10;i++)
		{
			// Set position of 4 axes
			res = SetAxis(value+00, iInterface, HID_USAGE_X);
			res = SetAxis(value+10, iInterface, HID_USAGE_Y);
			res = SetAxis(value+20, iInterface, HID_USAGE_Z);
			res = SetAxis(value+30, iInterface, HID_USAGE_RX);
			res = SetAxis(value+40, iInterface, HID_USAGE_RZ);

			// Press Button 1, Keep button 3 not pressed
			res = SetBtn(TRUE, iInterface, 1);
			res = SetBtn(FALSE, iInterface, 3);
		}

	Sleep(20);
	value+=10;
	}
```

**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:

## *Detecting Changes*

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:

Where ChangedCB is the user-defined callback function and hDlg is the handle to the application's top dialog box.

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

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. This is why in this example it **posts** a message to the application's top dialog box (passed as parameter *data*) and returns.

- 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 *data* always points to the data registered as second parameter of function *RegisterRemovalCB*.

## *Receptor Unit*

To take advantage of vJoy ability to process **Force Feedback** (FFB) data, you need to add a **receptor** unit to the feeder.

```
typedef struct _FFB_DATA {
	ULONG  size;        (DevID);
	ULONG  cmd;
	UCHAR  *data;
} FFB_DATA;
{
	_tprintf(L"Failed to start FFB on vJoy device number %d.\n", DevID);
	goto Exit;
}
else
	_tprintf(L"Started FFB on vJoy device number %d - OK\n", DevID);

// Register FFB callback function
// Function to register: FfbFunction1
// User Data: Device ID
FfbRegisterGenCB(FfbFunction1, &DevID);
```

The receptor unit receives the FFB data from a **source application**, and processes the FFB data. The data can be passed on to another entity (e.g. a physical joystick) or processed in place.

The Receptor is activated by **Acquiring** one or more vJoy devices (if not acquired yet), then **Starting** the devices' FFB capabilities and finally **registering** a single user-defined FFB callback function.

Once registered, the user-defined FFB callback function is called by a vJoy device every time a new FFB packet arrives from the **source application**. This function is called in the application thread and is **blocking**. This means that you must return from the FFB callback function ASAP – never wait in this function for the next FFB packet!

The SDK offers you a wide range of FFB helper-functions to process the FFB packet and a demo application that demonstrates the usage of the helper-functions. The helper-functions are efficient and can be used inside the FFB callback function.

Start a vJoy device' FFB capabilities by calling function *FfbStart()*.
Register a  user-defined FFB callback function by calling *FfbRegisterGenCB()*.

The FFB callback function is defined by the user. The function interface is as follows:

Where *FfbFunction1* is the name of the user-defined callback function. Parameter *FfbPacket* is a data packet (Type FFB_DATA) arriving from the vJoy device. Parameter *userdata* is a pointer to a user-defined buffer. You are not required to understand the structure of the FFB_DATA structure – just pass it to the the various FFB helper-functions.

**Structure FFB_DATA:**

Normally, you are not required to understand this structure as it is usually passed to the various helper function. However, you might want to access the raw FFB packet.

FFB_DATA Fields:

**size**: Size of FFB_DATA structure in bytes
**cmd**: Reserved
**data**: Array of size-8 bytes holding the FFB packet.

**FFB Helper Functions:**

These functions receive a pointer to FFB_DATA as their first parameter and return a DWORD status. The returned value is either `ERROR_SUCCESS` on success or other values on failure.

Use these functions to analyze the FFB data packets avoiding direct access to the raw FFB_DATA structure.

## *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.

```
VJOYINTERFACE_API BOOL    __cdecl    vJoyEnabled(void);
```
Returns TRUE if vJoy version 2.x is installed and enabled.

```
VJOYINTERFACE_API SHORT __cdecl     GetvJoyVersion(void);
```
Return the version number of the installed vJoy. To be used only after `vJoyEnabled()`

```
VJOYINTERFACE_API PVOID   __cdecl    GetvJoyProductString(void);
VJOYINTERFACE_API PVOID   __cdecl    GetvJoyManufacturerString(void);
VJOYINTERFACE_API PVOID   __cdecl    GetvJoySerialNumberString(void);
```
These functions return an LPTSTR that points to the correct data (Product, Manufacturer or Serial number). To be used only after `vJoyEnabled()`

```
VJOYINTERFACE_API BOOL    __cdecl    DriverMatch(WORD * DllVer, WORD * DrvVer);
```
Returns TRUE if vJoyInterface.dll file version and vJoy Driver version are identical. Otherwise returns FALSE.
Optional (You may pass NULL):
Output parameter *DllVer*: If a pointer to WORD is passed then the value of the **DLL file** version will be written to this parameter (e.g. 0x215).
Output parameter *DrvVer*: If a pointer to WORD is passed then the value of the **Driver** version will be written to this parameter (e.g. 0x215).

```
VJOYINTERFACE_API VOID    __cdecl    RegisterRemovalCB((CALLBACK *)(BOOL, BOOL,
PVOID)ConfChangedCB, PVOID * UserData);
```
This function registers a user-defined **ConfChangedCB** callback fuction that is called everytime a vJoy device is added or removed.
Paremeter *ConfChangedCB* is a pointer to the user-defined callback function.
Parameter *UserData* is a pointer to a user-defined data item. The callback function recieves this pointer as its third parameter.
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.

```
VJOYINTERFACE_API enum VjdStat      __cdecl    GetVJDStatus(UINT 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

[NEW]

```
VJOYINTERFACE_API BOOL    __cdecl   isVJDExists(UINT rID);
```
Returns TRUE if the specified device exists (Configured and enabled).
Returns FALSE otherwise (Including the following cases: Device does not exist, disabled, driver not installed)

[NEW]
```
VJOYINTERFACE_API int     __cdecl   GetOwnerPid(UINT rID);
```
Returns the Process ID (PID) of the process that owns the specified device.
If the device is owned by a process, then the function returns a positive integer which is the PID of the owner.
Otherwise, the function returns one of the following negative numbers:
NO_FILE_EXIST (-13): Usually indicates a FREE device (No owner)
NO_DEV_EXIST (-12): Usually indicates a MISSING device
BAD_DEV_STAT (-11): Indicates some internal problem

```
VJOYINTERFACE_API BOOL        __cdecl   AcquireVJD(UINT 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.

```
VJOYINTERFACE_API VOID        __cdecl   RelinquishVJD(UINT 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.

```
VJOYINTERFACE_API BOOL        __cdecl   UpdateVJD(UINT rID, PVOID pData);
```
Update the position data of the specified device.
Use only after device has been successfully acquired.
Input parameter is a pointer to structure of type JOYSTICK_POSITION 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.

```
VJOYINTERFACE_API int     __cdecl   GetVJDButtonNumber(UINT rID);
```
If function succeeds,  returns the number of buttons in the specified device. Valid values are 0 to 128
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

```
VJOYINTERFACE_API int     __cdecl   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  (from version 2.0.1)

```
VJOYINTERFACE_API int     __cdecl   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  ( from version 2.0.1)

```
VJOYINTERFACE_API BOOL    __cdecl  GetVJDAxisExist(UINT rID, UINT Axis);
```

Returns TRUE is the specified axis exists in the specified device
**Axis** values can be:

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

```
VJOYINTERFACE_API BOOL    __cdecl  ResetVJD(UINT 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).

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

```
VJOYINTERFACE_API BOOL __cdecl  ResetButtons(UINT rID);
```
Resets all buttons (To 0) in the specified device.

```
VJOYINTERFACE_API BOOL __cdecl  ResetPovs(UINT rID);
```
Resets all POV Switches (To -1) in the specified device.

```
VJOYINTERFACE_API BOOL __cdecl  SetAxis(LONG Value, UINT rID, UINT Axis);
```
Write Value to a given axis defined in the specified VDJ.
**Value** in the range 0x1-0x8000
**Axis** can be one of the following:

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

```
VJOYINTERFACE_API BOOL    __cdecl  SetBtn(BOOL Value, UINT rID, UCHAR nBtn);
```
Write **Value** (TRUE or FALSE) to a given button defined in the specified VDJ.
**nBtn** can in the range 1-128

```
VJOYINTERFACE_API BOOL    __cdecl  SetDiscPov(int Value, UINT rID, UCHAR 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

```
VJOYINTERFACE_API BOOL    __cdecl  SetContPov(DWORD Value,UINT rID,UCHAR 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

## *FFB Functions*

The following functions are used for accessing and manipulating Force Feedback data.

```
VJOYINTERFACE_API VOID    __cdecl    FfbRegisterGenCB(FfbGenCB cb, PVOID data);
```
Register a FFB callback function that will be called by the driver every time a FFB data packet arrives. For additional information see Receptor Unit section.

```
VJOYINTERFACE_API BOOL    __cdecl    FfbStart(UINT rID);
```
Enable the FFB mechanism of the specified VDJ.
Return TRUE on success. Otherwise return FALSE.

```
VJOYINTERFACE_API VOID    __cdecl    FfbStop(UINT rID);
```
Disable the FFB mechanism of the specified VDJ.

[NEW]
```
VJOYINTERFACE_API BOOL    __cdecl    IsDeviceFfb(UINT rID);
```
Return TRUE if specified device supports FFB. Otherwise return FALSE.

[NEW]
```
VJOYINTERFACE_API BOOL    __cdecl    IsDeviceFfbEffect(UINT rID, UINT Effect)
```
Return TRUE if specified device supports a specific FFB Effect. Otherwise return FALSE.
The FFB Effect is indicated by its Usage.
List of effect Usages:

| | |
|---|---|
| HID_USAGE_CONST (0x26): | Usage ET Constant Force |
| HID_USAGE_RAMP (0x27): | Usage ET Ramp |
| HID_USAGE_SQUR (0x30): | Usage ET Square |
| HID_USAGE_SINE (0x31): | Usage ET Sine |
| HID_USAGE_TRNG (0x32): | Usage ET Triangle |
| HID_USAGE_STUP (0x33): | Usage ET Sawtooth Up |
| HID_USAGE_STDN (0x34): | Usage ET Sawtooth Down |
| HID_USAGE_SPRNG (0x40): | Usage ET Spring |
| HID_USAGE_DMPR (0x41): | Usage ET Damper |
| HID_USAGE_INRT (0x42): | Usage ET Inertia |
| HID_USAGE_FRIC (0x43): | Usage ET Friction |

## *FFB Helper Functions*

```
VJOYINTERFACE_API DWORD    __cdecl    Ffb_h_DeviceID(const FFB_DATA *Packet, int *DeviceID);
```
Get the origin of the FFB data packet.
If valid device ID was found then returns ERROR_SUCCESS and sets the ID (Range 1-15) in **DeviceID**.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. DeviceID is undefined.
If Packet is malformed or Device ID is out of range then returns ERROR_INVALID_DATA. DeviceID is undefined.

```
VJOYINTERFACE_API DWORD    __cdecl Ffb_h_Type(const FFB_DATA * Packet, FFBPType *Type);
```
Get the type of the FFB data packet.

**Type** may be one of the following:

```
// Write
PT_EFFREP          // Usage Set Effect Report
PT_ENVREP          // Usage Set Envelope Report
PT_CONDREP         // Usage Set Condition Report
PT_PRIDREP         // Usage Set Periodic Report
PT_CONSTREP        // Usage Set Constant Force Report
PT_RAMPREP         // Usage Set Ramp Force Report
PT_CSTMREP         // Usage Custom Force Data Report
PT_SMPLREP         // Usage Download Force Sample
PT_EFOPREP         // Usage Effect Operation Report
PT_BLKFRREP        // Usage PID Block Free Report
PT_CTRLREP         // Usage PID Device Control
PT_GAINREP         // Usage Device Gain Report
PT_SETCREP         // Usage Set Custom Force Report

// Feature
PT_NEWEFREP        // Usage Create New Effect Report
PT_BLKLDREP        // Usage Block Load Report
PT_POOLREP         // Usage PID Pool Report
```

If valid Type was found then returns ERROR_SUCCESS and sets **Type**.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Feature is undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Feature is undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Packet(const FFB_DATA * Packet, WORD *Type, int *DataSize, BYTE *Data[]);
```

Extract the raw FFB data packet and the command type (Write/Set Feature).

If valid Packet was found then returns ERROR_SUCCESS and -
Sets **Type** to IOCTRL value (Expected values are IOCTL_HID_WRITE_REPORT and IOCTL_HID_SET_FEATURE).
Sets **DataSize** to the size (in bytes) of the payload data (FFB_DATA.data ).
Sets **Data** to the payload data (FFB_DATA.data ) - this is an array of bytes.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_EBI(const FFB_DATA * Packet, int *Index);
```

Get the Effect Block Index

If valid Packet was found then returns ERROR_SUCCESS and sets **Index** to the value of Effect Block Index (if applicable). Expected value is '1'.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed or does not contain an Effect Block Index then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Const(const FFB_DATA * Packet, FFB_EFF_CONST*  Effect);
```

Get parameters of an Effect of type Constant (PT_EFFREP)

Effect structure (FFB_EFF_CONST) definition:

```
typedef struct _FFB_EFF_CONST {
      BYTE        EffectBlockIndex; // Usually 1
      FFBEType    EffectType;       // ET_CONST(1)
      WORD        Duration;         // Value in milliseconds. 0xFFFF means infinite
```

```
	WORD		TrigerRpt;
	WORD		SamplePrd;
	BYTE		Gain;
	BYTE		TrigerBtn;
	BOOL		Polar;			// How to interpret force direction Polar (0-360°)
						// or Cartesian (X,Y)
	union
	{
		BYTE	Direction;		// Polar direction: (0x00-0xFF correspond to 0-360°)
		BYTE	DirX;			// X direction:
						// Positive values are To the right of the centre (X);
						// Negative are Two's complement
	};
	BYTE		DirY;			// Y direction:
						// Positive values are below the centre (Y);
						// Negative are Two's complement
} FFB_EFF_CONST;
```

If Constant Effect Packet was found then returns ERROR_SUCCESS and fills structure **Effect**
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Ramp(const FFB_DATA * Packet,
FFB_EFF_RAMP*  RampEffect);
```

Get parameters of an Effect of type Ramp (`PT_RAMPREP`)

Effect structure (`FFB_EFF_RAMP`) definition:

```
typedef struct _FFB_EFF_RAMP {
	BYTE		EffectBlockIndex; // Usually 1
	BYTE		Start;			// The Normalized magnitude at the start of the effect
	BYTE		End;			// The Normalized magnitude at the end of the effect
} FFB_EFF_RAMP;
```

If Ramp effect Packet was found then returns ERROR_SUCCESS and fills structure Effect.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_EffOp(const FFB_DATA * Packet,
FFB_EFF_OP*  Operation);
```

Get parameters of an Effect of type Operation (`PT_EFOPREP`) that describe the effect operation (Start/Solo/Stop) and loop count.

Effect structure (`FFB_EFF_OP`) definition:

```
typedef struct _FFB_EFF_OP {
	BYTE		EffectBlockIndex; // Usually 1
	FFBOP		EffectOp;		// Operation (EFF_START(1)/EFF_SOLO(2)/EFF_STOP(3))
	BYTE		LoopCount;		// Number of repetitions
} FFB_EFF_OP;
```

If Operation Effect Packet was found then returns ERROR_SUCCESS and fills structure Operation- this structure holds Effect Block Index, Operation(Start, Start Solo, Stop) and Loop Count.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Period(const FFB_DATA * Packet,
FFB_EFF_PERIOD*  Effect);
```

Get parameters of an Effect of type Periodic (`PT_PRIDREP`) that describe the periodic attribute of an effect.

Effect structure (FFB_EFF_PERIOD) definition:

```
typedef struct _FFB_EFF_PERIOD {
	BYTE		EffectBlockIndex; // Usually 1
	BYTE		Magnitude;
	BYTE		Offset;
	BYTE		Phase;
	WORD		Period;
} FFB_EFF_PERIOD;
```

If Periodic Packet was found then returns ERROR_SUCCESS and fills structure Effect – this structure holds Effect Block Index, Magnitude, Offset, Phase and period.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Cond(const FFB_DATA * Packet,
FFB_EFF_COND*  Condition);
```

Get parameters of an Effect of type Conditional (PT_CONDREP).

Effect structure (FFB_EFF_COND) definition:

```
typedef struct _FFB_EFF_COND {
	BYTE		EffectBlockIndex; // Usually 1
	BOOL		isY;
	BYTE		CenterPointOffset;// CP Offset: Range 10000 to 10000
	BYTE		PosCoeff;	// Positive Coefficient: Range  10000 to 10000
	BYTE		NegCoeff;	// Negative Coefficient: Range  10000 to 10000
	BYTE		PosSatur;	// Positive Saturation: Range 0 - 10000
	BYTE		NegSatur;	// Negative Saturation: Range 0 - 10000
	BYTE		DeadBand;	// Dead Band: : Range 0 - 10000
} FFB_EFF_COND;
```

If Condition Packet was found then returns ERROR_SUCCESS and fills structure Condition - this structure holds Effect Block Index, Direction (X/Y), Centre Point Offset, Dead Band and other conditions.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Envlp(const FFB_DATA * Packet,
FFB_EFF_ENVLP*  Envelope);
```

Get parameters of an Effect of type Envelope (PT_ENVREP).

Effect structure (FFB_EFF_ENVLP) definition:

```
typedef struct _FFB_EFF_ENVLP {
	BYTE		EffectBlockIndex;
	BYTE		AttackLevel;
	BYTE		FadeLevel;
	WORD		AttackTime;
	WORD		FadeTime;
} FFB_EFF_ENVLP;
```

If Envelope Packet was found then returns ERROR_SUCCESS and fills structure Envelope
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_EffNew(const FFB_DATA * Packet,
FFBEType * Effect);
```

Get the type of the next effect. Parameter **Effect** can get one of the following values:

```
	ET_NONE	=	0	//	No Force
	ET_CONST	=	1	//	Constant Force
```

```
ET_RAMP      =      2    //    Ramp
ET_SQR       =      3    //    Square
ET_SINE      =      4    //    Sine
ET_TRNGL     =      5    //    Triangle
ET_STUP      =      6    //    Sawtooth Up
ET_STDN      =      7    //    Sawtooth Down
ET_SPRNG     =      8    //    Spring
ET_DMPR      =      9    //    Damper
ET_INRT      =      10   //    Inertia
ET_FRCTN     =      11   //    Friction
ET_CSTM      =      12   //    Custom Force Data
```

If valid Packet was found then returns ERROR_SUCCESS and sets the new **Effect** type
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

[NEW]

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Constant(const FFB_DATA * Packet, FFB_EFF_CONSTANT * ConstantEffect);
```

Get parameters of an Effect of type Constant (`PT_CONSTREP`).
If Constant Packet was found then returns ERROR_SUCCESS and fills structure ConstantEffect
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_DevCtrl(const FFB_DATA * Packet, FFB_CTRL *  Control);
```

Get device-wide control instructions. **Control** can get one of the following values:

```
CTRL_ENACT        = 1   // Enable all device actuators.
CTRL_DISACT       = 2   // Disable all the device actuators.
CTRL_STOPALL      = 3   // Stop All Effects Issues a stop on every running effect.
CTRL_DEVRST       = 4   // Device Reset
                        // Clears any device paused condition,
                        // enables all actuators and clears all effects from memory.
CTRL_DEVPAUSE     = 5   // Device Pause
                        // All effects on the device are paused
                        // at the current time step.
CTRL_DEVCONT      = 6   // Device Continue
                        // All effects that running when the
                        // device was paused are restarted from their last time step.
```

```
VJOYINTERFACE_API DWORD __cdecl Ffb_h_DevGain(const FFB_DATA * Packet,  BYTE * Gain);
```

Get device Global gain in parameter **Gain**.

If valid Packet was found then returns ERROR_SUCCESS and gets the device global gain.
If Packet is NULL then returns ERROR_INVALID_PARAMETER. Output parameters are undefined.
If Packet is malformed then returns ERROR_INVALID_DATA. Output parameters are undefined.

### *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 x64 target machines.

When you deploy your feeder, don't forget to supply the user with file vJoyInterface.dll of the correct bitness.
Location of Feeder
You may locate your feeder anywhere you like provided that file **vJoyInterface.dll** is on the feeder's search path.
Here are a few points that may help you decide where to deploy your feeder:

1. If you choose to link to file **vJoyInterface.dll** provided by this SDK you risk to use a non-optimal library. If the user upgrades vJoy, you risk linking to an outdated library.
2. If you choose to link to file **vJoyInterface.dll** provided by vJoy Driver installation you need to locate the library file while installing your feeder.

### *Location of vJoyInterface.dll*

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:<br>Location of vJoy driver installer and uninstaller |
| DllX64Location | C:\Program Files\vJoy\x64 | • Location of 64-bit utilities and libraries<br>• Only on 64-bit Machines |
| DllX86Location | C:\Program Files\vJoy\x86 | • Location of 32-bit utilities and libraries<br>• 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*

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.

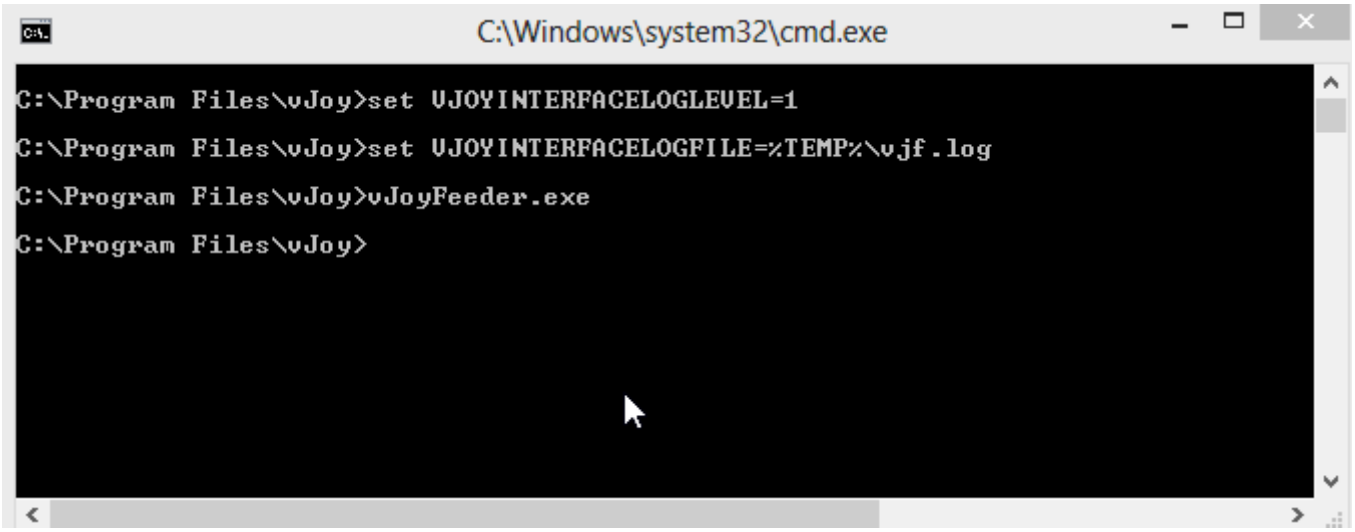
This feature is intended both for helping you develop your feeder and to collect data at the user's location – provided the user is willing to trigger logging for you. By default, logging state is OFF.

## *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-2.1.5\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.