# How to write a vJoy Feeder (C/C++)

*Updated: 19-Oct-2015 (v2.1.6)*

# Feeder Overview

A vJoy feeder enables you to feed one or more vJoy devices with position data and optionally to receive Force Feedback (FFB) data from the vJoy device.
Try to write a simple as possible a feeder:

**Device**
A feeder can feed as many as 16 vJoy device and to select the device to be fed.
However, in many cases, you can safely assume that the vJoy device you intend to feed is device number One.
In this case, you the feeder will just have to verify that the device exists, and you can eliminate the vJoy device detection and selection logic.

**Device Removal/Insertion**
The feeder may be designed to detect a change in the vJoy device status. It can react to removal of a vJoy device and to introducing of a device.
In most cases these capabilities are not needed because the user is not expected to make changes while using vJoy.

**FFB Support**
This feature complicates the feeder. If your target application (Simulator, game etc.) does not support FFB or if your hardware does not support FFB – don't implement it.

**Efficiency vs Better code**
Feeding the vJoy device can be made using a low-level interface function (**UpdateVJD**) that updates an entire device at once or using a set of high level interface functions each updating a single vJoy device control such as a button or an axis.
The former approach is more efficient than the latter one.
Using the latter approach will result in a simpler code and is less sensitive to future changes in the API.
The vJoy high-level interface functions are quite efficient and unless a large number of controls are expected to change simultaneously it is recommended to use it.
Use the low-level interface function only in cases such as a racing wheel scenario when the user may simultaneously turn the wheel (X-Axis), press Accelerator pedal (Rx Axis), press the Brakes pedal (Ry Axis) and press a few buttons.

## *API*

All access to vJoy driver and to the vJoy devices is done through vJoy interface functions that are implemented in file vJoyInterface.dll.
It is advisable to base your feeder on 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). 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. |

In addition, the feeder may include an **FFB receptor** that receives FFB data from the target application. The receptor is implemented as a callback function that treats the FFB data as quickly as possible and returns.
The API offers a wide range of FFB helper-functions for analysis of the FFB data packets.

### *Feeding the vJoy device*

#### Test vJoy Driver:

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

```
// Get the driver attributes (Vendor ID, Product ID, Version Number)
if (!vJoyEnabled())
{
	_tprintf("Failed Getting vJoy attributes.\n");
	return -2;
}
else
{
	_tprintf("Vendor: %S\nProduct :%S\nVersion Number:%S\n",\
	TEXT(GetvJoyManufacturerString()),\
	TEXT(GetvJoyProductString()),\
	TEXT(GetvJoySerialNumberString()));
};

// Test interface DLL matches vJoy driver
// Compare versions
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);
```

### Test vJoy Virtual Devices:

Check which devices are installed and what their state is:

```
	// Get the state of the requested device (iInterface)
	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\n\
	                                    Cannot continue\n", iInterface);
	return -3;
	case VJD_STAT_MISS:
	_tprintf("vJoy Device %d is not installed or disabled\n\
	                                    Cannot continue\n", iInterface);
	return -4;
	default:
	_tprintf("vJoy Device %d general error\nCannot continue\n", iInterface);
		return -1;
	};
```

### Acquire the vJoy Device:

Until now the feeder 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):

```
// Acquire the target if not already owned
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);
```

### Feed vJoy Device:

The time has come to do some real work: feed the vJoy device with position data.
Reset the device once then send the position data for every control (axis, button,POV) at a time.

```
// Reset this device to default values
ResetVJD(iInterface);

// Feed the device in endless loop
while(1)
{
	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;
}
```

### Relinquish the vJoy Device:

You must relinquish the device when the driver exits:

```
RelinquishVJD(iInterface);
```

## *Force Feedback support*

To take advantage of vJoy ability to process Force Feedback (FFB) data, you need to add a receptor unit to the feeder.
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 yet acquired) and registering a 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.
Register a user-defined FFB callback function by calling **FfbRegisterGenCB**().

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

# Software Reference

## Interface Functions

### *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.

| | |
|---|---|
| **GetvJoyVersion** | Get the vJoy driver Version Number |
| **GetvJoyProductString** | Get string describing vJoy driver |
| **GetvJoyManufacturerString** | Get string describing manufacturer of vJoy driver |
| **GetvJoySerialNumberString** | Get string describing serial number (version) of vJoy driver |
| **vJoyEnabled** | Checks if at least one vJoy Device is enabled |
| **DriverMatch** | Checks matching of vJoy Interface DLL file with driver |
| **RegisterRemovalCB** | Register a Callback function that is called when a vJoy device is added or removed |
| **ConfChangedCB** | An application-defined callback function registered by function **RegisterRemovalCB** |

# GetvJoyVersion function

Get the vJoy driver Version Number.

## Syntax

```
C++
      VJOYINTERFACE_API SHORT __cdecl  GetvJoyVersion(void);
```

## Parameters

This function has no parameters.

## Return Value

Driver version number if available. Otherwise returnes 0.

## Remarks

The output of this function is interprated as a hexadecimal value where the lower 3 nibbles hold the version number.
For example, version 2.1.6 will be returned as 0x0216.

# GetvJoyProductString function

Get string describing vJoy driver

## Syntax

```
C++
      VJOYINTERFACE_API PVOID    __cdecl    GetvJoyProductString(void);
```

## Parameters

This function has no parameters.

## Return Value

Driver product string if available. Otherwise returns NULL.

## Remarks

The pointer has to be cast into PWSTR
Currently, value is L"**vJoy - Virtual Joystick**"

# GetvJoyManufacturerString function

Get string describing manufacturer of vJoy driver

## Syntax

```
C++
      VJOYINTERFACE_API PVOID    __cdecl    GetvJoyProductString(void);
```

## Parameters

This function has no parameters.

## Return Value

Driver manufacturer string if available. Otherwise returns NULL.

## Remarks

The pointer has to be cast into PWSTR
Currently, value is L"**Shaul Eizikovich**"

# GetvJoySerialNumberString function

Get string describing serial number (version) of vJoy driver

## Syntax

```
C++
      VJOYINTERFACE_API PVOID    __cdecl GetvJoySerialNumberString(void);
```

## Parameters

This function has no parameters.

## Return Value

Driver Serial number string if available. Otherwise returns NULL.

## Remarks

The pointer has to be cast into PWSTR
Value is  of the type L"**2.1.6**"

# vJoyEnabled function

Checks if at least one vJoy Device is enabled

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl  vJoyEnabled(void);
```

## Parameters

This function has no parameters.

## Return Value

TRUE if vJoy Driver is installed and there is at least one enabled vJoy device.

# DriverMatch function

Checks matching of vJoy Interface DLL file with driver

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl DriverMatch(
            WORD * DllVer,
            WORD * DrvVer
      );
```

## Parameters

*DllVer [opt out]*
: Pointer to DLL file version number.

*DrvVer [opt out]*
: Pointer to Driver version number.

## Return Value

Returns TRUE if vJoyInterface.dll file version and vJoy Driver version are identical. Otherwise returns FALSE.

## Remarks

Use this function to verify DLL/Driver compatibility.
If a valid pointer to an output buffer is passed to parameter *DllVer* – function **DriverMatch** will set the buffer to the version value of file vJoyInterface.dll (e.g. 0X0216).
If a valid pointer to an output buffer is passed to parameter *DrvVer* – function **DriverMatch** will set the buffer to the version value of the installed vJoy driver (e.g. 0X0205).
Valid pointers may be used by the feeder for version comparison or to display to the user. If you don't intend to use these values you may set the parameters to NULL.
Function **DriverMatch** returns TRUE only if vJoyInterface.dll file version and vJoy Driver version are identical.

# RegisterRemovalCB function

Register a Callback function that is called when a vJoy device is added or removed

## Syntax

C++

```
VJOYINTERFACE_API VOID __cdecl RegisterRemovalCB(
      (CALLBACK *)(BOOL, BOOL, PVOID)ConfChangedCB,
      PVOID * UserData
);
```

## Parameters

*ConfChangedCB [in]*

Pointer to the application-defined callback function.

*UserData [opt in]*

Pointer to the application-defined data item.

## Return Value

This function does not return a value.

## Remarks

Function **RegisterRemovalCB** registers a application-defined **ConfChangedCB** callback function that is called every time a vJoy device is added or removed.
This is useful if you need your feeder to be aware of configuration changes that are introduced while it is running.
**ConfChangedCB** callback function is a placeholder for a user defined function that the user should freely name.
**ConfChangedCB** callback function received the pointer to UserData, the application-defined data item, as its third parameter.

## Example

```
// Example of registration of callback function
// The callback function is named vJoyConfChangedCB
// When function vJoyConfChangedCB will be called - its 3rd parameter will be hDlg
      RegisterRemovalCB(vJoyConfChangedCB, (PVOID)hDlg);
```

# ConfChangedCB callback function

An application-defined callback function registered by function **RegisterRemovalCB**. Called when a vJoy device is added or removed.
**ConfChangedCB** is a placeholder for the application-defined function name.

## Syntax

C++

```
VJOYINTERFACE_API void CALLBACK ConfChangedCB(
      BOOL Removed,
      BOOL First,
      PVOID data
);
```

## Parameters

*Removed [in]*
Removal/Addition of vJoy Device.
*First [opt in]*
First device to be Removed/Added
*data [opt inout]*
Pointer to the application-defined data item.

## Return Value

This function does not return a value.

## Remarks

Register your callback function using function **RegisterRemovalCB** when you want your feeder to be alerted when a vJoy device is added or removed.
You may give your callback function any name you wish.
Your callback function must return as quickly as possible since it is executed in the computer's system context. Refraining from a quick return may prevent the addition or removal of the device.
Some actions may be taken only on removal of first vJoy device (such as stopping the feeder) while some actions are to be carried out on any removal/addition.
Use combination of parameters (Remover/First) to determine the exact situation. There is no way to detect the **last** removal/addition of device.

## Example

```
// Definition of callback function
// The function posts a message when called and immediately returns
void CALLBACK vJoyConfChangedCB(BOOL Removed, BOOL First, PVOID data)
{
	HWND hDlg = (HWND)data;
	PostMessage(hDlg, WM_VJOYCHANGED, (WPARAM)Removed, (LPARAM)First);
}


// Handler for message WM_VJOYCHANGED.
// Called every time a vJoy device is added or removed

	switch (message)
	{
	case WM_VJOYCHANGED:
		if (wParam && lParam) // First remove message
			vJoyDeviceRemoved();
		else if (!wParam) // Any arrival message
			vJoyDeviceArrived();
		break;
```

### *Device Information*

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).

| | |
|---|---|
| **GetVJDButtonNumber** | Get the number of buttons |
| **GetVJDDiscPovNumber** | Get the number of Discrete POV Hat switches |
| **GetVJDContPovNumber** | Get the number of Continuous POV Hat switches |
| **GetVJDAxisExist** | Check if a specific axis exists |
| **GetVJDStatus** | Get Status of a vJoy device |

# GetVJDButtonNumber function

Get the number of buttons

## Syntax

```
C++
      VJOYINTERFACE_API int __cdecl    GetVJDButtonNumber(
            UINT rID
      );
```

## Parameters

*rID [in]*

ID of vJoy device.

## Return Value

Number of buttons configured for the vJoy device defined by rID. Valid range is 0-128.
In case that the function fails to get the correct number of buttons, the function returns a negative value as follows:

| | |
|---|---|
| `BAD_PREPARSED_DATA (-2):` | Function failed to get device's pre-parsed data. |
| `NO_CAPS (-3):` | Function failed to get device's capabilities. |
| `BAD_N_BTN_CAPS (-4):` | Function failed to get the “Number of Buttons” field in the device's capabilities structure. |
| `BAD_BTN_CAPS (-6):` | Function failed to extract the Button Capabilities from the device's capabilities structure. |
| `BAD_BTN_RANGE (-7):` | Function failed to extract the Button Range from device's capabilities structure. |

## Remarks

The **GetVJDButtonNumber** function queries the number of buttons assigned for a specific vJoy device as indicated by parameter rID. Any positive number in the range, including 0 is a valid value. Negative values mean that there is either a problem with the device or that it does not exist.

# GetVJDDiscPovNumber function

Get the number of Discrete POV Hat switches

## Syntax

```
C++
     VJOYINTERFACE_API int __cdecl    GetVJDDiscPovNumber(
          UINT rID
     );
```

## Parameters

*rID [in]*

ID of vJoy device.

## Return Value

Number of Discrete POV Hat switches configured for the vJoy device defined by rID. Valid range is 0-4.
In case that the function fails to get the correct number of switches, the function returns 0.

## Remarks

The **GetVJDDiscPovNumber** function queries the number of Discrete POV Hat switches assigned for a specific vJoy device as indicated by parameter rID. Any positive number in the range, including 0 is a valid value.
The result 0 may indicate both a failure or 0 switches.
Discrete POV Hat switches have 5 states: North, West, South, East and neutral.

# GetVJDContPovNumber function

Get the number of Continuous POV Hat switches

## Syntax

```
C++
      VJOYINTERFACE_API int __cdecl    GetVJDContPovNumber(
            UINT rID
      );
```

## Parameters

*rID [in]*

ID of vJoy device.

## Return Value

Number of Continuous POV Hat switches configured for the vJoy device defined by rID. Valid range is 0-4.
In case that the function fails to get the correct number of switches, the function returns 0.

## Remarks

The **GetVJDDiscPovNumber** function queries the number of Continuous POV Hat switches assigned for a specific vJoy device as indicated by parameter rID. Any positive number in the range, including 0 is a valid value.
The result 0 may indicate both a failure or 0 switches.
Continuous POV Hat switches have many states reflecting all possible positions and in addition a neutral state.

# GetVJDAxisExist function

Check if a specific axis exists.

## Syntax

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

## Parameters

*rID [in]*
ID of vJoy device.
*Axis [in]*
Axis Number

## Return Value

TRUE if the axis exists in the given vJoy Device.
FALSE otherwise.

## Remarks

The **GetVJDAxisExist** function queries if a given axis exists for a specific vJoy device as indicated by parameter rID.
Every one of the axes that may be assigned to a device is defined by a number as documented in the USB documentations and in header file public.h
Possible values are:

| Axis | Macro definition | Value |
|---|---|---|
| X | HID_USAGE_X | 0x30 |
| Y | HID_USAGE_Y | 0x31 |
| Z | HID_USAGE_Z | 0x32 |
| Rx | HID_USAGE_RX | 0x33 |
| Ry | HID_USAGE_RY | 0x34 |
| Rz | HID_USAGE_RZ | 0x35 |
| Slider0 | HID_USAGE_SL0 | 0x36 |
| Slider1 | HID_USAGE_SL1 | 0x37 |
| Wheel | HID_USAGE_WHL | 0x38 |
| POV | HID_USAGE_POV | 0x39 |

# GetVJDStatus function

Get Status of a vJoy device.

## Syntax

```
C++
      VJOYINTERFACE_API enum VjdStat __cdecl GetVJDStatus(
            UINT rID
      );
```

## Parameters

*rID [in]*

ID of vJoy device.

## Return Value

Status of the vJoy device. See Remarks for interpretation of the status.

## Remarks

Every vJoy device is attributed a status. According to the status the feeder should Acquire, Relinquish, start or stop feeding the device with data or report a problem.
The possible statuses are:

| | |
|---|---|
| **VJD_STAT_OWN** | The vJoy Device is owned by this feeder. |
| **VJD_STAT_FREE** | The vJoy Device is NOT owned by any feeder (including this one). |
| **VJD_STAT_BUSY** | The vJoy Device is owned by another feeder. 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 |

There are a few options to change the state of a vJoy device:

| | |
|---|---|
| **VJD_STAT_OWN → VJD_STAT_FREE** | By calling function **RelinquishVJD**. |
| **VJD_STAT_ FREE → VJD_STAT_OWN** | By calling function **AcquireVJD**. |
| **VJD_STAT_BUSY → VJD_STAT_FREE** | By forcing the owner of the device (another feeder) to relinquish the device. |
| **VJD_STAT_MISS → VJD_STAT_FREE** | By adding this device (Use application vJoyConf). |

### *Device Feeding*

The following functions are used for the purpose of changing a vJoy Device's position. In other words, to load new values into its controls (Buttons, Axes and POV Hat switches).

| | |
|---|---|
| **AcquireVJD** | Acquire a vJoy device by the feeder |
| **RelinquishVJD** | Relinquish an acquired vJoy device by the feeder |
| **UpdateVJD** | Set the positions of a vJoy device controls |
| **ResetVJD** | Reset all controls to their default values |
| **ResetAll** | Reset all controls to their default values on all vJoy devices |
| **ResetButtons** | Reset all buttons to their default values |
| **ResetPovs** | Reset all POV hat switches to their default values |
| **SetAxis** | Set an axis to its desired position |
| **SetBtn** | Set a button to its desired position |
| **SetDiscPov** | Set a discrete POV Hat Switch to its desired position |
| **SetContPov** | Set a continuous POV Hat Switch to its desired position |

# AcquireVJD function

Acquire a vJoy device by the feeder.

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl  AcquireVJD(
            UINT rID
      );
```

## Parameters

*rID [in]*
ID of vJoy device.

## Return Value

TRUE if the vJoy device has been successfully acquired by the feeder.
FALSE otherwise.

## Remarks

The  feeder must call AcquireVJD function before it can start feeding the vJoy device with data.
The feeder should call **RelinquishVJD** so that another feeder may acquire the vJoy device when the specified vJoy Device is no longer required.
Additional calls to this function are ignored.

# RelinquishVJD function

Relinquish an acquired vJoy device by the feeder.

## Syntax

```
C++
      VJOYINTERFACE_API VOID __cdecl  RelinquishVJD(
            UINT rID
      );
```

## Parameters

*rID [in]*

ID of vJoy device.

## Return Value

This function does not return a value.

## Remarks

The  feeder should call **RelinquishVJD** function in order to make the vJoy device, previously acquired by the feeder, available to other feeders.
If a vJoy device is not relinquished, other feeders cannot acquire the device.
Function **RelinquishVJD** should be called only if the vJoy device has been previously acquired using function **AcquireVJD**.
Additional calls to **RelinquishVJD** will be ignored.

# UpdateVJD function

Set the positions of a vJoy device controls.

## Syntax

```
C++
     VJOYINTERFACE_API BOOL __cdecl  UpdateVJD(
          UINT rID,
          PVOID pData
     );
```

## Parameters

*rID [in]*
: ID of vJoy device.

pData [in]
: Pointer to position data

## Return Value

TRUE if the feeder succeeded writing data to the vJoy device.
FALSE otherwise.

## Remarks

Function **UpdateVJD** sets the positions of a vJoy device controls. Controls are the Buttons, Axes and POV Hat Switches.
Function **UpdateVJD** may be called only after the device has been **acquired** and **owned**.
The pointer to position data, pData, points to a valid structure **JOYSTICK_POSITION_V2** defined in header file **public.h**.
**Note**: This is a low level function. As consequence it is the most efficient method to load position data onto a vJoy device. On the other hand, this function is not opaque to future changes in the driver architecture.
High level functions such as **SetAxis**, **SetBtn**, **SetDiscPov** and **SetContPov** are less efficient because they call **UpdateVJD** function. However, they are opaque to future changes in changes in the driver architecture. Also, using them makes your code more readable.

# ResetVJD function

Reset all controls to their default values

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl  ResetVJD(
            UINT rID
      );
```

## Parameters

*rID [in]*

ID of vJoy device.

## Return Value

TRUE if the feeder succeeded to reset the controls.
FALSE otherwise.

## Remarks

It is advisable to call function **ResetVJD** right after the acquisition of a vJoy device. This will place all device's controls in their respective default positions.
The default positions are determined by a combination of hard-coded positions and and registry entries.
In the lack of overriding registry entries, the default positions are as follows:

| | |
|---|---|
| Axes X,Y,Z | Middle Point |
| All other Axes | 0 |
| POV Hat Switches | Neutral (-1) |
| Buttons | Not pressed (0) |

# ResetAll function

Reset all controls to their default values on all vJoy devices.

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl  ResetAll(void);
```

## Parameters

This function has no parameters.

## Return Value

TRUE if the feeder succeeded to reset the controls.
FALSE otherwise.

## Remarks

For details see **ResetVJD**.

# ResetButtons function

Reset all buttons to their default values

## Syntax

```
C++
     VJOYINTERFACE_API BOOL __cdecl  ResetButtons(
          UINT rID
     );
```

## Parameters

*rID [in]*
: ID of vJoy device.

## Return Value

TRUE if the feeder succeeded to reset the controls.
FALSE otherwise.

## Remarks

Function **ResetButtons** will place all device's buttons in their respective default positions.
The default positions are determined by a combination of hard-coded positions and and registry entries.
In the lack of overriding registry entries, the buttons are by default unpressed.

# ResetPovs function

Reset all POV hat switches to their default values

## Syntax

```
C++
     VJOYINTERFACE_API BOOL __cdecl  ResetPovs(
          UINT rID
     );
```

## Parameters

*rID [in]*
ID of vJoy device.

## Return Value

TRUE if the feeder succeeded to reset the controls.
FALSE otherwise.

## Remarks

Function **ResetPovs** will place all device's POV hat switches in their respective default positions.
The default positions are determined by a combination of hard-coded positions and and registry entries.
In the lack of overriding registry entries, the switches are by default in their neutral position.

# SetAxis function

Set an axis to its desired position

## Syntax

```C++
VJOYINTERFACE_API BOOL __cdecl  SetAxis(
      LONG Value,
      UINT rID,
      UINT Axis
);
```

## Parameters

*Value [in]*
Position of the target axis. Range 0x0001-0x8000
*rID [in]*
ID of vJoy device.
*Axis [in]*
Target axis

## Return Value

TRUE if the feeder succeeded to set the target axis.
FALSE otherwise.

## Remarks

Function **SetAxis** will set *Axis* in vJoy device *rID* to *Value*.
The possible axis *value* range is 0x0001to 0x8000 (32768).
The target *axis* may be one of the following:

| Axis | Macro definition | Value |
|---|---|---|
| X | HID_USAGE_X | 0x30 |
| Y | HID_USAGE_Y | 0x31 |
| Z | HID_USAGE_Z | 0x32 |
| Rx | HID_USAGE_RX | 0x33 |
| Ry | HID_USAGE_RY | 0x34 |
| Rz | HID_USAGE_RZ | 0x35 |
| Slider0 | HID_USAGE_SL0 | 0x36 |
| Slider1 | HID_USAGE_SL1 | 0x37 |
| Wheel | HID_USAGE_WHL | 0x38 |
| POV | HID_USAGE_POV | 0x39 |

Function **SetAxis** may be called only after the device has been **acquired** and **owned**.

**Note**: This is a high level function that calls Function **UpdateVJD**. As consequence it is not the most efficient method to load position data onto a vJoy device. On the other hand, this function is  opaque to future changes in the driver architecture.
Low level function **UpdateVJD** is a more efficient function. However, it is not opaque to future changes in the driver architecture.

# SetBtn function

Set a button to its desired position

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl SetBtn(
            BOOL Value,
            UINT rID,
            UCHAR nBtn
      );
```

## Parameters

*Value [in]*
Set/Unset
*rID [in]*
ID of vJoy device.
*nBtn [in]*
Target button

## Return Value

TRUE if the feeder succeeded to set the target button.
FALSE otherwise.

## Remarks

Function **SetBtn** will set/unset a single button in vJoy device *rID*.
The target *button* may be in the range: 1-128.
Function **SetBtn** may be called only after the device has been **acquired** and **owned**.
**Note**: This is a high level function that calls Function **UpdateVJD**. As consequence it is not the most efficient method to load position data onto a vJoy device. On the other hand, this function is opaque to future changes in the driver architecture.
Low level function **UpdateVJD** is a more efficient function. However, it is not opaque to future changes in the driver architecture.

# SetDiscPov function

Set a discrete POV Hat Switch to its desired position

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl SetDiscPov(
            int Value,
            UINT rID,
            UCHAR nPov
      );
```

## Parameters

*Value [in]*
Desired position
*rID [in]*
ID of vJoy device.
*nPov [in]*
Target POV Hat Switch

## Return Value

TRUE if the feeder succeeded to set the target POV Hat switch.
FALSE otherwise.

## Remarks

Function **SetDiscPov** will set a single POV Hat switch in vJoy device rID to its desired position.
The target POV Hat Switch nPov may be in the range: 1-4.
The desired position, Value, can be set to one of the following values:

**0** North (or Forwards)
**1** East (or Right)
**2** South (or backwards)
**3** West (or left)
**-1** Neutral (Nothing pressed)

Function **SetDiscPov** may be called only after the device has been **acquired** and **owned**.
**Note**: This is a high level function that calls Function **UpdateVJD**. As consequence it is not the most efficient method to load position data onto a vJoy device. On the other hand, this function is opaque to future changes in the driver architecture.
Low level function **UpdateVJD** is a more efficient function. However, it is not opaque to future changes in the driver architecture.

# SetContPov function

Set a continuous POV Hat Switch to its desired position

## Syntax

```
C++
      VJOYINTERFACE_API BOOL __cdecl SetContPov(
            DWORD Value,
            UINT rID,
            UCHAR nPov
      );
```

## Parameters

*Value [in]*
Desired position
*rID [in]*
ID of vJoy device.
*nPov [in]*
Target POV Hat Switch

## Return Value

TRUE if the feeder succeeded to set the target POV Hat switch.
FALSE otherwise.

## Remarks

Function **SetContPov** will set a single POV Hat switch in vJoy device *rID* to its desired position.
The target POV Hat Switch *nPov* may be in the range: 1-4.
The desired position, *Value*, can take a value in the range 0-35999 or -1.
Value -1 represents the neutral state of the POV Hat Switch.
The range 0-35999 represents its position in 1/100 degree units, where 0 signifies North (or forwards), 9000 signifies East (or right), 18000 signifies South (or backwards), 27000 signifies West (or left) and so forth.
Function **SetContPov** may be called only after the device has been **acquired** and **owned**.
**Note**: This is a high level function that calls Function **UpdateVJD**. As consequence it is not the most efficient method to load position data onto a vJoy device. On the other hand, this function is opaque to future changes in the driver architecture.
Low level function **UpdateVJD** is a more efficient function. However, it is not opaque to future changes in the driver architecture.

### *Force Feedback*

The following functions are used to write a Force Feedback (FFB) receptor unit.

| | |
|---|---|
| **FfbCB callback function** | Callback function that is called every time a source application sends FFB data to a vJoy device. |
| **FfbRegisterGenCB** | Register a Callback function that is called when a source application sends FFB data to a vJoy device. |
| **Ffb_h_DeviceID** | Extract information from FFB data packet : ID of the vJoy device of origin. |
| **Ffb_h_Type** | Extract information from FFB data packet : Type of the data packet. |
| **Ffb_h_EBI** | Extract information from FFB data packet : Effect Block Index of the data packet. |
| **Ffb_h_Eff_Report** | Extract information from FFB data packet of type **Effect Report**. |
| **Ffb_h_Eff_Ramp** | Extract information from FFB data packet of type **Ramp Effect** |
| **Ffb_h_EffOp** | Extract information from **operation** FFB data packet |
| **Ffb_h_DevCtrl** | Extract information from device-wide control instructions FFB data packet |
| **Ffb_h_Eff_Period** | Extract information from FFB data packet : Parameters of a periodic effect. |
| **Ffb_h_Eff_Cond** | Extract information from FFB data packet : Parameters of a condition block. |
| **Ffb_h_DevGain** | Extract information from FFB data packet : Device Global gain. |
| **Ffb_h_Eff_Envlp** | Extract information from FFB data packet : Effect Envelope block. |
| **Ffb_h_EffNew** | Extract information from FFB data packet : Type of the next effect. |
| **Ffb_h_Eff_Constant** | Extract information from FFB data packet : Magnitude of a constant force effect. |

# FfbCB callback function

Callback function that is called every time a source application sends FFB data to a vJoy device.

## Syntax

C++

```
VJOYINTERFACE_API VOID __cdecl FfbCB(
      PVOID FfbPacket,
      PVOID data
);
```

## Parameters

*FfbPacket [in]*

Pointer to the FFB data packet.

*data [opt in]*

Pointer to the application-defined data item.

## Return Value

This function does not return a value.

## Remarks

Register your callback function using function **FfbRegisterGenCB** so that application-defined **FfbCB** callback function will be called every time a source application sends FFB data to a vJoy device.
**FfbCB** callback function is a placeholder for a user defined function that the user should freely name.
**FfbCB** callback function received the pointer to FFB data packet and the application-defined data item, as its 2nd parameter.
The data packet is opaque. Pass it to FFB helper functions in order to analyze it.
Your callback function must return as quickly as possible since it is executed in the source application's context. Refraining from a quick return will block the execution of the source application.

## Example

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

// An example of a simple FFB callback functional
// This function is called with every FFB data packet emitted by the source app
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
	if (ERROR_SUCCESS == Ffb_h_EBI((FFB_DATA *)data, &BlockIndex))
		_tprintf("\n > Effect Block Index: %d", BlockIndex);
}
```

# FfbRegisterGenCB function

Register a Callback function that is called when a source application sends FFB data to a vJoy device.

## Syntax

```
C++
      VJOYINTERFACE_API VOID __cdecl FfbRegisterGenCB(
            FfbGenCB cb,
            PVOID data
      );
```

## Parameters

*cb [in]*

Pointer to the application-defined callback function.

*data [opt in]*

Pointer to the application-defined data item.

## Return Value

This function does not return a value.

## Remarks

Function **FfbRegisterGenCB** registers a application-defined **Ffbcb** callback function that is called every time a source application sends FFB data to a vJoy device.
A **Ffbcb** callback function must be registered in order to establish a functional **receptor**.
**Ffbcb** callback function is a placeholder for a user defined function that the user should freely name.
**Ffbcb** callback function received the pointer to *data,* the application-defined data item, as its 2nd parameter.

## Example

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

# Ffb_h_DeviceID function

Extract information from FFB data packet : ID of the vJoy device of origin.

## Syntax

```
C++
      VJOYINTERFACE_API DWORD __cdecl Ffb_h_DeviceID(
            const FFB_DATA * Packet,
            int *DeviceID
      );
```

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*DeviceID [out]*

Pointer to vJoy device ID.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_DeviceID** analyzes an FFB data packet. If the data is valid then parameter *DeviceID* receives the ID of the vJoy device of origin and the function returns ERROR_SUCCESS. Valid values are 1 to 15.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL. *DeviceID* is undefined.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *DeviceID* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       /////// Packet Device ID
       int DeviceID;
       TCHAR  TypeStr[100];
       if (ERROR_SUCCESS == Ffb_h_DeviceID((FFB_DATA *)data, &DeviceID))
              _tprintf("\n > Device ID: %d", DeviceID);
}
```

# Ffb_h_Type function

Extract information from FFB data packet : Type of the data packet.

## Syntax

```
C++
      VJOYINTERFACE_API DWORD __cdecl Ffb_h_Type(
            const FFB_DATA * Packet,
            FFBPType *Type
      );
```

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Type [out]*

Pointer to the Type of FFB data packet.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Type** analyzes an FFB data packet. If the data is valid then parameter *Type* receives the type of the data packet and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Type* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       FFBPType      Type;
       TCHAR  TypeStr[100];

       if (ERROR_SUCCESS == Ffb_h_Type((FFB_DATA *)data, &Type))
       {
              if (!PacketType2Str(Type, TypeStr))
                     _tprintf("\n > Packet Type: %d", Type);
              else
                     _tprintf("\n > Packet Type: %s", TypeStr);

       }
}
```

# Ffb_h_EBI function

Extract information from FFB data packet : Effect Block Index of the data packet.

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Index [out]*

Pointer to the effect block index.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_EBI** analyzes an FFB data packet. If the data is valid then parameter *Index* receives the effect block index of the data packet (usually '1') and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Index* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       int BlockIndex;

       if (ERROR_SUCCESS == Ffb_h_EBI((FFB_DATA *)data, &BlockIndex))
             _tprintf("\n > Effect Block Index: %d", BlockIndex);
```

# Ffb_h_Eff_Report function

Extract information from FFB data packet of type **Effect Report**.

## Syntax

```
C++
      VJOYINTERFACE_API DWORD __cdecl Ffb_h_Eff_Report(
            const FFB_DATA * Packet,
            FFB_EFF_REPORT *  Effect
      );
```

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Effect [out]*

Pointer to the structure that holds effect report data.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Eff_Report** analyzes an FFB data packet. If the data is valid then parameter *Effect* receives the structure holding the effect report data and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Effect* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       FFB_EFF_CONST Effect;
       if (ERROR_SUCCESS == Ffb_h_Eff_Report((FFB_DATA *)data, &Effect))
       {      // The effect report is OK
              // Analyze the effect direction
              if (Effect.Polar)
                     _tprintf("\n >> Direction: %d deg (%02x)",\
                     Polar2Deg(Effect.Direction), Effect.Direction);
              else
              {
                     _tprintf("\n >> X Direction: %02x", Effect.DirX);
                     _tprintf("\n >> Y Direction: %02x", Effect.DirY);
              };
       };
}
```

# Ffb_h_Eff_Ramp function

Extract information from FFB data packet of type **Ramp Effect**

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*RampEffect [out]*

Pointer to the structure that holds Ramp effect data.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Eff_Ramp** analyzes an FFB data packet. If the data is valid then parameter *RampEffect* receives the structure holding the effect data and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *RampEffect* is undefined.

The Ramp Effect Data describes the effect as follows:

- Effect Block Index — Usually 1
- Start — Magnitude of at the beginning of the effect
- End — Magnitude of at the end of the effect

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       FFB_EFF_RAMP RampEffect;
       if (ERROR_SUCCESS == Ffb_h_Eff_Ramp((FFB_DATA *)data, &RampEffect))
       {
              _tprintf("\n >> Ramp Start: %d", RampEffect.Start);
              _tprintf("\n >> Ramp End: %d", RampEffect.End);
       };

}
```

# Ffb_h_EffOp function

Extract information from **operation** FFB data packet

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Operation [out]*

Pointer to the structure that holds effect operation data.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_EffOp** analyzes an FFB data packet. If the data is valid then parameter *Operation* receives the structure holding the effect data and the function returns ERROR_SUCCESS.
An operation is one of the followings Start/Solo/Stop and may also define the number of repetitions.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Operation* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
	FFB_EFF_OP	Operation;
	TCHAR	EffOpStr[100];
	if (ERROR_SUCCESS == Ffb_h_EffOp((FFB_DATA *)data, &Operation))
	{
		// Conver the operation to string: Start, Stop or Solo
		EffectOpStr(Operation.EffectOp, EffOpStr);
		// Print the operation
		_tprintf("\n >> Effect Operation: %s", EffOpStr);

		// Print the number of repetitions
		if (Operation.LoopCount == 0xFF)
			_tprintf("\n >> Loop until stopped");
		else
			_tprintf("\n >> Loop %d times", \
			static_cast<int>(Operation.LoopCount));

	};

}
```

# Ffb_h_DevCtrl function

Extract information from device-wide control instructions FFB data packet

## Syntax

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

## Parameters

*Packet [in]*
Pointer to a FFB data packet.
*Control [out]*
Pointer to the structure that holds control data.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_DevCtrl** analyzes an FFB data packet. If the data is valid then parameter *Control* receives the structure holding the vJoy device data and the function returns ERROR_SUCCESS.
A **control** is one of the following values:

| | |
|---|---|
| CTRL_ENACT | Enable all device actuators. |
| CTRL_DISACT | Disable all the device actuators. |
| CTRL_STOPALL | Stop All Effects. Issues a stop on every running effect. |
| CTRL_DEVRST | Device Reset.<br>Clears any device paused condition, enables all actuators and clears all effects from memory. |
| CTRL_DEVPAUSE | Device Pause.<br>All effects on the device are paused at the current time step. |
| CTRL_DEVCONT | Device Continue<br>All effects that running when the device was paused are restarted from their last time step. |

Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Control* is undefined.

# Ffb_h_Eff_Period function

Extract information from FFB data packet : Parameters of a periodic effect.

## Syntax

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

## Parameters

*Packet [in]*
: Pointer to a FFB data packet.

*Effect [out]*
: Pointer to the structure that holds periodic effect data.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Eff_Period** analyzes an FFB data packet. If the data is valid then parameter Effect receives the structure holding the attributes of the periodic effect and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. Effect is undefined.
Periodic Effects are Sine-wave, square-wave, saw-tooth and a few others. They have periodic attributes which are extracted using **Ffb_h_Eff_Period**.
These attributes are:

- Magnitude The amplitude of the wave.
- Offset The up/down shift of the wave pattern
- Phase The shift of the wave pattern in the temporal axis
- Period The wave period

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
	FFB_EFF_PERIOD EffPrd;
	if (ERROR_SUCCESS == Ffb_h_Eff_Period((FFB_DATA *)data, &EffPrd))
	{
		_tprintf(L"\n >> Magnitude: %d", EffPrd.Magnitude );
		_tprintf(L"\n >> Offset: %d",\
		TwosCompWord2Int(static_cast<WORD>(EffPrd.Offset)));
		_tprintf(L"\n >> Phase: %d", EffPrd.Phase);
		_tprintf(L"\n >> Period: %d", static_cast<int>(EffPrd.Period));
	};
}
```

# Ffb_h_Eff_Cond function

Extract information from FFB data packet : Parameters of a condition block.

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Condition [out]*

Pointer to the structure that holds condition block  data.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Eff_Cond** analyzes an FFB data packet. If the data is valid then parameter *Condition* receives the structure holding the attributes of the condition block and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. condition is undefined.
Condition blocks describe spring, damper, Inertia and friction effects. Note that there is a condition block for every force direction (Usually x and y).

The condition block parameters are:

- Center Point Offset
- Positive Coefficient
- Negative Coefficient
- Positive Saturation
- Negative  Saturation
- Dead Band
- Direction (X or Y)

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
	FFB_EFF_COND Condition;
	if (ERROR_SUCCESS == Ffb_h_Eff_Cond((FFB_DATA *)data, &Condition))
	{
		// Get the direction (X/Y) of this condition block
		if (Condition.isY)
			_tprintf(L"\n >> Y Axis");
		else
			_tprintf(L"\n >> X Axis");


		// Get condition parameters for this direction
		_tprintf(L"\n >> Center Point Offset: %d",\
	TwosCompWord2Int((WORD)Condition.CenterPointOffset));
		_tprintf(L"\n >> Positive Coefficient: %d",\
	TwosCompWord2Int((WORD)Condition.PosCoeff));
		_tprintf(L"\n >> Negative Coefficient: %d",\
	TwosCompWord2Int((WORD)Condition.NegCoeff));
		_tprintf(L"\n >> Positive Saturation: %d", Condition.PosSatur);
		_tprintf(L"\n >> Negative Saturation: %d", Condition.NegSatur);
		_tprintf(L"\n >> Dead Band: %d", Condition.DeadBand);
	};
}
```

# Ffb_h_DevGain function

Extract information from FFB data packet : Device Global gain.

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Gain [out]*

Pointer to the structure that holds  Device Global gain.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_DevGain** analyzes an FFB data packet. If the data is valid then parameter *Gain* receives the global gain of the device and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Gain* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       BYTE Gain;

       // The gain range: 0 to 0xFF ( Equivalent to 0%-100%)
       if (ERROR_SUCCESS == Ffb_h_DevGain((FFB_DATA *)data, &Gain))
              _tprintf(L"\n >> Global Device Gain: %d", Byte2Percent(Gain));
}
```

# Ffb_h_Eff_Envlp function

Extract information from FFB data packet : Effect Envelope block.

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*Envelope [out]*

Pointer to the structure that holds the envelope block parameters.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Eff_Envlp** analyzes an FFB data packet. If the data is valid then parameter *Envelope* receives the the parameters of the envelope block and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Envelope* is undefined.

The Envelope block modifies some of the parameters of the corresponding effect:
Attack Level, Attack Time, Fade Level and Attack Level.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       FFB_EFF_ENVLP Envelope;
       if (ERROR_SUCCESS == Ffb_h_Eff_Envlp((FFB_DATA *)data, &Envelope))
       {
       _tprintf(L"\n >> Attack Level: %d", TwosCompWord2Int((WORD)Envelope.AttackLevel));
       _tprintf(L"\n >> Fade Level: %d", TwosCompWord2Int((WORD)Envelope.FadeLevel));
       _tprintf(L"\n >> Attack Time: %d", static_cast<int>(Envelope.AttackTime));
       _tprintf(L"\n >> Fade Time: %d", static_cast<int>(Envelope.FadeTime));
       };
}
```

# Ffb_h_EffNew function

Extract information from FFB data packet : Type of the next effect.

## Syntax

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

## Parameters

*Packet [in]*
: Pointer to a FFB data packet.

*Effect [out]*
: Pointer to the structure that holds the type of the next effect.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_EffNew** analyzes an FFB data packet. If the data is valid then parameter *Effect* receives the the Type of the next FFB effect and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *Effect* is undefined.

# Ffb_h_Eff_Constant function

## Syntax

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

## Parameters

*Packet [in]*

Pointer to a FFB data packet.

*ConstantEffect [out]*

Pointer to the structure that holds magnitude of the constant force.

## Return Value

This function returns error code. See remarks for details.

## Remarks

Function **Ffb_h_Eff_Envlp** analyzes an FFB data packet. If the data is valid then parameter *ConstantEffect* receives the the parameters of the envelope block and the function returns ERROR_SUCCESS.
Other possible return values:
ERROR_INVALID_PARAMETER: Data packet is NULL.
ERROR_INVALID_DATA: Malformed Data packet or ID out of range. *ConstantEffect* is undefined.

## Example

```
// FFB callback function
void CALLBACK FfbFunction1(PVOID data, PVOID userdata)
{
       FFB_EFF_CONSTANT ConstantEffect;
       if (ERROR_SUCCESS == Ffb_h_Eff_Constant((FFB_DATA *)data, &ConstantEffect))
              _tprintf(L"\n >> Constant Magnitude: %d",\
       TwosCompWord2Int((WORD)ConstantEffect.Magnitude));
}
```

## Interface Structures

- **JOYSTICK_POSITION_V2**
- **FFB_EFF_REPORT**
- **FFB_EFF_RAMP**
- **FFB_EFF_OP**
- **FFB_EFF_PERIOD**
- **FFB_EFF_COND**
- **FFB_EFF_ENVLP**
- **FFB_EFF_CONSTANT**

# JOYSTICK_POSITION_V2 Structure

The **JOYSTICK_POSITION_V2** structure contains information about the joystick position, point-of-view position, and button state.

## Syntax

C++

```
typedef struct _JOYSTICK_POSITION_V2
{
        BYTE   bDevice;
        LONG   wThrottle;
        LONG   wRudder;
        LONG   wAileron;
        LONG   wAxisX;
        LONG   wAxisY;
        LONG   wAxisZ;
        LONG   wAxisXRot;
        LONG   wAxisYRot;
        LONG   wAxisZRot;
        LONG   wSlider;
        LONG   wDial;
        LONG   wWheel;
        LONG   wAxisVX;
        LONG   wAxisVY;
        LONG   wAxisVZ;
        LONG   wAxisVBRX;
        LONG   wAxisVBRY;
        LONG   wAxisVBRZ;
        LONG   lButtons;
        DWORD  bHats;
        DWORD  bHatsEx1;
        DWORD  bHatsEx2;
        DWORD  bHatsEx3;
        LONG lButtonsEx1;
        LONG lButtonsEx2;
        LONG lButtonsEx3;
} JOYSTICK_POSITION_V2, *PJOYSTICK_POSITION_V2;
```

## Members

bDevice
: Index of device.
Range 1-16.

wThrottle
: Reserved.

wRudder
: Reserved.

wAileron
: Reserved.

**wAxisX**
: X-Axis.

**wAxisY**
: Y-Axis

**wAxisZ**
: Z-Axis.

**wAxisXRot**
: Rx-Axis.

**wAxisYRot**
: Ry-Axis.

**wAxisZRot**
: Rz-Axis.

**wSlider**
: Slider0-Axis.

**wDial**
: Slider1-Axis.

**wWheel**
: Reserved.

**wAxisVX**
: Reserved.

**wAxisVY**
: Reserved.

**wAxisVZ**
: Reserved.

**wAxisVBRX**
Reserved.

**wAxisVBRY**
Reserved.

**wAxisVBRZ**
Reserved.

**lButtons**
Buttons 1-32.

**bHats**
POV Hat Switch
If device set to continuous switches – this is the value of POV Hat Switch #1
If device set to discrete switches – every nibble represents a POV Hat Switch.

**bHatsEx1**
POV Hat Switch
If device set to continuous switches – this is the value of POV Hat Switch #2
If device set to discrete switches – not used.

**bHatsEx2**
POV Hat Switch
If device set to continuous switches – this is the value of POV Hat Switch #3
If device set to discrete switches – not used.

**bHatsEx3**
POV Hat Switch
If device set to continuous switches – this is the value of POV Hat Switch #4
If device set to discrete switches – not used.

lButtonsEx1
Buttons 33-64.

lButtonsEx2
Buttons 65-96.

lButtonsEx3
Buttons 97-128.

## Remarks

**Axis members**

Valid value for **Axis** members are in range 0x0001 – 0x8000.

**Button members**

Valid value for **Button** members are in range 0x00000000 (all 32 buttons are unset) to 0xFFFFFFFF (all  buttons are set). The least-significant-bit representing the lower-number button (e.g. button #1).

**POV Hat Switch members**

The interpretation of these members depends on the configuration of the vJoy device.
**Continuous**: Valid value for POV Hat Switch member is either 0xFFFFFFFF (neutral) or in the range of 0 to 35999 .
**Discrete**: Only member **bHats** is used. The lowest nibble is used for switch #1, the second nibble for switch #2, the third nibble for switch #3 and the highest nibble for switch #4.
Each nibble supports one of the following values:

| | |
|---|---|
| 0x0 | North (forward) |
| 0x1 | East (right) |
| 0x2 | South (backwards) |
| 0x3 | West (Left) |
| 0xF | Neutral |

# FFB_EFF_REPORT Structure

The **FFB_EFF_REPORT** structure contains general information about the FFB effect.

## Syntax

C++

```
typedef struct _FFB_EFF_REPORT {
        BYTE          EffectBlockIndex;
        FFBType       EffectType;
        WORD          Duration;
        WORD          TrigerRpt;
        WORD          SamplePrd;
        BYTE          Gain;
        BYTE          TrigerBtn;
        BOOL          Polar;
        union
        {
                BYTE   Direction;
                BYTE   DirX;
        };
        BYTE          DirY;
} FFB_EFF_REPORT, *PFFB_EFF_REPORT;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

EffectType

The type of the effect.
For full list look in the definition of **FFBType**.

Duration

The duration of the effect (in milliseconds).
0xFFFF means infinite.

TriggerRpt

Trigger repeat.
0xFFFF means infinite.

**SamplePrd**

Sample Period
0xFFFF means infinite.

**TriggerBtn**

Reserved.

**Polar**

True:  Force direction Polar (0-360°)

False: Force direction Cartesian (X,Y)

**Direction**

If Force Direction is Polar: Range 0x00-0xFF corresponds to 0°-360°

**DirX**

If  Force Direction Cartesian:
X direction -Positive values are To the right of the center (X); Negative are Two's complement

**DirY**

If  Force Direction Cartesian:
Y direction -Positive values are To the below of the center (Y); Negative are Two's complement

## Remarks

This data packet is central to the definition of an effect. It holds all of the basic effect parameters such as type of effect, Duration and direction.
Other data packets may modify the data by adding Envelope, Condition et cetera.

# FFB_EFF_RAMP Structure

The **FFB_EFF_REPORT** structure contains general information about the FFB effect.

## Syntax

C++

```
typedef struct _FFB_EFF_RAMP {
        BYTE          EffectBlockIndex;
        LONG          Start;
        LONG          End;
} FFB_EFF_RAMP, *PFFB_EFF_RAMP;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

Start

The Normalized magnitude at the start of the effect.
Range -10000 to 10000

End

The Normalized magnitude at the end of the effect.
Range -10000 to 10000

## Remarks

This data packet modifies Ramp effect.

# FFB_EFF_OP Structure

The **FFB_EFF_OP** structure contains general information about the FFB effect.

## Syntax

```
C++
        typedef struct _FFB_EFF_OP {
                BYTE          EffectBlockIndex;
                FFBOP         EffectOp;
                BYTE          LoopCount;
        } FFB_EFF_OP, *PFFB_EFF_OP;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

EffectOp

Operation to apply on effect marked by **EffectBlockIndex**
Possible Operations are: Start, Solo, Stop

LoopCount

Number of times to loop. Stop not required.
0xFF means loop forever (until explicitly stopped).

## Remarks

This data packet Starts/Stops an FFB effect.

# FFB_EFF_PERIOD Structure

The **FFB_EFF_PERIOD** structure contains information about a periodic FFB effect.

## Syntax

C++

```
typedef struct _FFB_EFF_PERIOD {
        BYTE          EffectBlockIndex;
        DWORD         Magnitude;
        LONG          Offset;
        DWORD         Phase;
        DWORD         Period;
} FFB_EFF_PERIOD, *PFFB_EFF_PERIOD;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

Magnitude

The amplitude of the periodic effect.
Range 0 to 10000

Offset

The effect offset on the magnitude axis (Y axis)
The range of forces generated by the effect will be (Offset - Magnitude) to (Offset + Magnitude).
Range -10000 to 10000

Phase

The effect offset of the wave on the temporal axis (X axis).
Range: 0 – 35999 (Units: 1/100 degree)

Period

The period of the effect.
Range 0-32767

## Remarks

All periodic effects share the above parameters.

# FFB_EFF_COND Structure

The **FFB_EFF_COND** structure contains information about an FFB effect condition.

## Syntax

C++

```
typedef struct _FFB_EFF_COND {
	BYTE		EffectBlockIndex;
	BOOL		isY;
	LONG		CenterPointOffset; // CP Offset:  Range -10000 - 10000
	LONG		PosCoeff; // Positive Coefficient: Range -10000 - 10000
	LONG		NegCoeff; // Negative Coefficient: Range -10000 - 10000
	DWORD		PosSatur; // Positive Saturation: Range 0 – 10000
	DWORD		NegSatur; // Negative Saturation: Range 0 – 10000
	LONG		DeadBand; // Dead Band: : Range 0 – 1000
} FFB_EFF_COND, *PFFB_EFF_COND;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

isY

A condition block is defined for each direction of the effect.
This parameter is TRUE if the block refers to axis Y.

CenterPointOffset

Offset from axis 0 position.
Range -10000 to 10000

PosCoeff

The Normalized coefficient constant on the positive side of the neutral position.
Range -10000 to 10000

NegCoeff

The Normalized coefficient constant on the negative side of the neutral position .
Range -10000 to 10000

PosSatur

The Normalized maximum positive force output.
Range 0 to 10000

NegSatur

The Normalized maximum negative force output.
Range 0 to 10000

DeadBand

The region around CP Offset where the condition is not active.
In other words, the condition is not active between (Offset – Dead Band) and (Offset + Dead Band).
Range 0-10000

## Remarks

The following effect types use this block:

- Spring
- Damper
- Inertia
- Friction

If the metric is **less** than CP Offset - Dead Band, then the resulting force is given by the following formula:

force = Negative Coefficient * (q - (CP Offset – Dead Band))

Similarly, if the metric is **greater** than CP Offset + Dead Band, then the resulting force is given by the following formula:

force = Positive Coefficient * (q - (CP Offset + Dead Band))

where **q** is a type-dependent metric:

- A **spring** condition uses axis position as the metric.
- A **damper** condition uses axis velocity as the metric.
- An **inertia** condition uses axis acceleration as the metric.

# FFB_EFF_ENVLP Structure

The **FFB_EFF_ENVLP** structure contains information about an FFB effect envelope modifier.

## Syntax

```
C++
	typedef struct _FFB_EFF_ENVLP {
		BYTE		EffectBlockIndex;
		DWORD		AttackLevel;
		DWORD		FadeLevel;
		DWORD		AttackTime;
		DWORD		FadeTime;
	} FFB_EFF_ENVLP, *PFFB_EFF_ENVLP;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

AttackLevel

Normalized amplitude for the start of the envelope, from the baseline.
Range 0 to 10000

FadeLevel

Normalized amplitude to end the envelope, from baseline.
Range 0 to 10000

AttackTime

The transition time to reach the sustain level.

FadeTime

The fade time to reach the fade level.

## Remarks

The Envelope Block describes the envelope to be used by an effect. Note that not all effect types use odifies FFB effect parametenvelopes. The envelope mers.
The following effects are optionally modified by an envelope block:

- Constant Force
- Ramp
- Square-wave
- Sine-wave
- Triangle wave
- Sawtooth up
- Sawtooth down

# FFB_EFF_CONSTANT Structure

The **FFB_EFF_CONSTANT** structure contains information about an FFB Constant Force effect.

## Syntax

C++

```
typedef struct _FFB_EFF_CONSTANT {
        BYTE EffectBlockIndex;
        LONG Magnitude;
} FFB_EFF_CONSTANT, *PFFB_EFF_CONSTANT;
```

## Members

EffectBlockIndex

Index of the effect.
All data packets related to a specific effect carry the same index.
Since there is usually one effect at a time – the index is usually '1'.

Magnitude

Magnitude of constant force.
Range -10000 to 10000

## Interface Constants

| | |
|---|---|
| **VjdStat** | The **vjdStat** enumeration type defines a list of possible vJoy device states. |
| **FFBPType** | The **FFBPType** enumeration type defines a list of possible FFB data packets. |
| **FFBOP** | The **FFBOP** enumeration type defines a list of possible FFB Effect operations. |
| **FFB_CTRL** | The **FFB_CTRL** enumeration type defines a list of possible FFB Effect operations. |
| **FFBEType** | The **FFBEType** enumeration type defines a list of possible FFB Effects. |

# VjdStat enumeration

The **vjdStat** enumeration type defines a list of possible vJoy device states.

## Syntax

C++

```
enum VjdStat {
	VJD_STAT_OWN,
	VJD_STAT_FREE,
	VJD_STAT_BUSY,
	VJD_STAT_MISS,
	VJD_STAT_UNKN
};
```

## Constants

**VJD_STAT_OWN**

The vJoy Device is owned by this feeder.

**VJD_STAT_FREE**

The vJoy Device is NOT owned by any feeder (including this one).

**VJD_STAT_BUSY**

The vJoy Device is owned by another feeder.  It cannot be acquired by this feeder.

**VJD_STAT_MISS**

The vJoy Device is missing. It either does not exist or the driver is down.

**VJD_STAT_UNKN**

Unknown (error)

# FFBPType enumeration

The **FFBPType** enumeration type defines a list of possible FFB data packets.

## Syntax

C++

```
enum FFBType {
	PT_EFFREP     =  HID_ID_EFFREP,
	PT_ENVREP     =  HID_ID_ENVREP,
	PT_CONDREP    =  HID_ID_CONDREP,
	PT_PRIDREP    =  HID_ID_PRIDREP,
	PT_CONSTREP   =  HID_ID_CONSTREP,
	PT_RAMPREP    =  HID_ID_RAMPREP,
	PT_CSTMREP    =  HID_ID_CSTMREP,
	PT_SMPLREP    =  HID_ID_SMPLREP,
	PT_EFOPREP    =  HID_ID_EFOPREP,
	PT_BLKFRREP   =  HID_ID_BLKFRREP,
	PT_CTRLREP    =  HID_ID_CTRLREP,
	PT_GAINREP    =  HID_ID_GAINREP,
	PT_SETCREP    =  HID_ID_SETCREP,
	PT_NEWEFREP   =  HID_ID_NEWEFREP+0x10,
	PT_BLKLDREP   =  HID_ID_BLKLDREP+0x10,
	PT_POOLREP    =  HID_ID_POOLREP+0x10,
};
```

## Constants

**PT_EFFREP**

The FFB data packet contains an **Effect** Report.

**PT_ENVREP**

The FFB data packet contains an **Envelope** Report.

**PT_CONDREP**

The FFB data packet contains an **Condition** Report.

**PT_PRIDREP**

The FFB data packet contains an **Periodic** Report.

**PT_CONSTREP**

The FFB data packet contains an **Constant** Force Report.

**PT_RAMPREP**

The FFB data packet contains an **Ramp** Force Report.

PT_CSTMREP

The FFB data packet contains an **Custom** Force Report. (Not supported by vJoy)

PT_**SMPLREP**

The FFB data packet contains an Custom Force **download** sample. (Not supported by vJoy).

PT_EFOPREP

The FFB data packet contains an Effect **Operation** report. Effect Operation report contains command (Start/Stop/Solo) and number of iterations.

PT_BLKFRREP

The FFB data packet contains a **Block Free** report. (Not supported by vJoy).

PT_CTRLREP

The FFB data packet contains a **PID Device Control**. (Not supported by vJoy).

PT_ **GAINREP**

The FFB data packet contains a **Device Gain** report. (Not supported by vJoy).

PT_**SETCREP**

The FFB data packet contains a **Custom Force** report. (Not supported by vJoy).

PT_NEWEFREP

The FFB data packet contains a **Create New** report. (Not supported by vJoy).

PT_BLKLDREP

The FFB data packet contains a **Block Load** report. (Not supported by vJoy).

PT_POOLREP

The FFB data packet contains a **PID POOL** report. (Not supported by vJoy).

# FFBOP enumeration

The **FFBOP** enumeration type defines a list of possible FFB Effect operations.

## Syntax

C++

```
enum FFBOP
{
	EFF_START	= 1,
	EFF_SOLO	= 2,
	EFF_STOP	= 3,
};
```

## Constants

**EFF_START**

Start effect.

**EFF_SOLO**

Start effect and stop all other effects.

**EFF_STOP**

Stop effect.

# FFB_CTRL enumeration

The **FFB_CTRL** enumeration type defines a list of possible FFB Effect operations.

## Syntax

```C++

enum FFB_CTRL
{
	CTRL_ENACT    = 1,
	CTRL_DISACT   = 2,
	CTRL_STOPALL  = 3,
	CTRL_DEVRST   = 4,
	CTRL_DEVPAUSE = 5,
	CTRL_DEVCONT  = 6,
};
```

## Constants

**CTRL_ENACT**
Enable all device actuators.

**CTRL_DISACT**
Disable all the device actuators.

**CTRL_STOPALL**
Stop All Effects.
Issues a stop on every running effect.

**CTRL_DEVRST**
Device Reset.
Clears any device paused condition, enables all actuators and clears all effects from memory.

**CTRL_DEVPAUSE**
Device Pause.
All effects on the device are paused at the current time step.

**CTRL_DEVCONT**
Device Continue.
All effects that are running when the device was paused are restarted from their last time step.

# FFBEType enumeration

The **FFBEType** enumeration type defines a list of possible FFB Effects.

## Syntax

```
C++

enum FFBEType // FFB Effect Type
{

	// Effect Type
	ET_NONE		=	0,
	ET_CONST	=	1,
	ET_RAMP		=	2,
	ET_SQR		=	3,
	ET_SINE		=	4,
	ET_TRNGL	=	5,
	ET_STUP		=	6,
	ET_STDN		=	7,
	ET_SPRNG	=	8,
	ET_DMPR		=	9,
	ET_INRT		=	10,
	ET_FRCTN	=	11,
	ET_CSTM		=	12,
};
```

## Constants

**ET_NONE**

No Force

**ET_CONST**

Constant Force

**ET_RAMP**

Ramp

**ET_SQR**

Square

**ET_SINE**

Sine

**ET_TRNGL**

Triangle

**ET_STUP**

Sawtooth Up

**ET_STDN**

Sawtooth Down

**ET_SPRNG**
: Spring

**ET_DMPR**
: Damper

**ET_INRT**
: Inertia

**ET_FRCTN**
: Friction

**ET_CSTM**
: Custom Force Data

# FfbGenCB function pointer

Application-defined callback function for the **FfbRegisterGenCB** function .

## Syntax

C++

```
typedef void (CALLBACK *FfbGenCB)(
       PVOID FfbPacket,
       PVOID data
);
```

## Parameters

*FfbPacket [in]*

Pointer to the FFB data packet.

*data [opt in]*

Pointer to the application-defined data item.

## Return Value

This function does not return a value.