How to write a vJoy Feeder

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

The feeder must link to interface assembly vJoyInterfaceWrap by applying the using directive:

```
using vJoyInterfaceWrap;
```

In order for this namespace to be available while compiling and linking the feeder, the following files should be placed in the same folder:
vJoyInterface.dll: The C-language API library
vJoyInterfaceWrap.dll: The C# wrapper around the C-language API library
The wrapper contains one class – vJoy. The feeder must contain a vJoy object:

```
joystick = new vJoy();
```

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. Obtain information about the driver. An installed driver implies at least one vJoy device.

	Test if driver matches interface DLL file
Test Virtual Device(s):	Get information regarding one or more devices. Read information about a specific device capabilities: Axes, buttons and POV hat switches.
Device acquisition:	Obtain status of a vJoy device. Acquire the device if the device status is owned or is free.
Updating:	Inject position data to a device (as long as the device is 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, create a vJoy object and check if the vJoy driver is installed and that it is what you expected:

```
joystick = new vJoy();

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

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

Test vJoy Virtual Devices:

Check which devices are installed and what their state is:

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

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

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):

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

Feed vJoy Device:

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

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

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

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

Relinquish the vJoy Device:

You must relinquish the device when the driver exits:

```
joystick.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**().

```
public FfbInterface(TesterForm dialog)
{
      dlg = dialog;
      joystick = dialog.joystick;
      // Start FFB Mechanism
      if (!joystick.FfbStart(id))
         throw new Exception(String.Format(
                  "Cannot start Forcefeedback on device {0}", id));

      // Convert Form to pointer and pass it as user data
      // to the callback function
      GCHandle h = GCHandle.Alloc(dialog);
      IntPtr parameter = (IntPtr)h;

      // Register the callback function & pass the dialog box object
      joystick.FfbRegisterGenCB(OnEffectObj, dialog);
}
```

Software Reference

Interface Functions

Interface Structures

Interface Constants

Interface Function Pointers

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#

```
      short GetvJoyVersion()
```

Parameters

This function has no parameters.

Return Value

Driver version number if available. Otherwise returns 0.

Remarks

The output of this function is interpreted 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#

```
      string GetvJoyProductString()
```

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#

```
      string GetvJoyManufacturerString()
```

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#

```
      string GetvJoySerialNumberString()
```

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#

```
bool vJoyEnabled()
```

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#

```
bool DriverMatch(
      ref UInt32 DllVer,
      ref UInt32 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#

```
void RegisterRemovalCB(
      RemovalCbFunc cb,
      object data
)
```

Parameters

cb [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

```
// Register callback function
// Function to register:     Removal()
// User data to pass:        label2
joystick.RegisterRemovalCB(Removal, label2);
```

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#

```
void ConfChangedCB(
     bool removal,
     bool first,
     object 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.

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#

```
int 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#

```
int 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#
      int   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#

```
bool GetVJDAxisExist(
      UInt32 rID,
      HID_USAGES 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#

```
VjdStat GetVJDStatus(
      UInt32 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#

```
bool AcquireVJD(
     UInt32 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#

```
void 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#
      bool UpdateVJD(
            UInt32 rID,
            ref JoystickState 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 **JoystickState** 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#

```
bool ResetVJD(
     UInt32 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#

```
bool ResetAll()
```

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#

```
bool ResetButtons(
      UInt32 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#

```
bool ResetPovs(
      UInt32 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#
      bool SetAxis(
            Int32 Value,
            UInt32 rID,
            HID_USAGES 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#

```
bool SetBtn(
      bool Value,
      UInt32 rID,
      uint 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#

```
SetDiscPov(
      Int32 Value,
      UInt32 rID,
      uint 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#

```
SetContPov(
      Int32 Value,
      UInt32 rID,
      uint 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#

```
void FfbCB(
      IntPtr FfbPacket,
      object 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: OnEffectObj
// User Data: dialog
joystick.FfbRegisterGenCB(OnEffectObj, dialog);

// An example of a simple FFB callback functional
// This function is called with every FFB data packet emitted by the source app
private static void OnEffectObj(IntPtr data, object userData)
{

     int DSize = 0;
     Byte[] arr = new Byte[20];
     UInt32 wType = 0;
     string TypeStr = "";

     // Get FFB packet raw data
     var result = joystick.Ffb_h_Packet(data, ref wType, ref DSize, ref arr);
     dlg.FfbTextBox_Write(\
             String.Format("\r\n\r\nFFB Packet size Size {0} ======", DSize + 8));
}
```

FfbRegisterGenCB function

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

Syntax

C#

```
void FfbRegisterGenCB(
      FfbCbFunc cb,
      object 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#

```
UInt32 Ffb_h_DeviceID(
      IntPtr  Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       /////// Packet Device ID
       int id = 0;
       // Get the ID of the device that this FFB packet refers to
       var result = joystick.Ffb_h_DeviceID(data, ref id);
       dlg.FfbTextBox_Write(String.Format("\r\n > Device ID: {0}", id));
}
```

Ffb_h_Type function

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

Syntax

C#

```
UInt32 Ffb_h_Type(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       FFBPType type = 0;
       string TypeStr = "";

       // Get the type of the packet
       var result = joystick.Ffb_h_Type(data, ref type);
       if (result == 0)
       {
        bool ok = PacketType2Str(type, ref TypeStr);
        if (!ok)
           dlg.FfbTextBox_Write(String.Format("\r\n > Packet Type: {0}", type));
        else
           dlg.FfbTextBox_Write(String.Format("\r\n > Packet Type: {0}", TypeStr));
       };
}
```

Ffb_h_EBI function

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

Syntax

C#

```
UInt32 Ffb_h_EBI(
      IntPtr Packet,
      ref Int32 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
private static void OnEffectObj(IntPtr data, object userData)
{
       // Get the index of the effect block (Always 1)
       int iBlock = 0;
       var  result = joystick.Ffb_h_EBI(data, ref iBlock);
       if (result == 0)
              dlg.FfbTextBox_Write(String.Format(\
                     "\r\n > Effect Block Index: {0}", iBlock));
}
```

Ffb_h_Eff_Report function

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

Syntax

C#

```
UInt32 Ffb_h_Eff_Report(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       // Effect Report
       vJoy.FFB_EFF_REPORT Effect = new vJoy.FFB_EFF_REPORT();
       var result = joystick.Ffb_h_Eff_Report(data, ref Effect);
       if (result == 0)
       {
        ///// This is an Effect Block
         // Effect type
         if (!EffectType2Str(Effect.EffectType, ref TypeStr))
             dlg.FfbTextBox_Write(String.Format("\r\n >> Effect Report: {0}", \
                                                Effect.EffectType));
         else
             dlg.FfbTextBox_Write(String.Format("\r\n >> Effect Report: {0}", TypeStr));

         // Effect Direction
         if (Effect.Polar)
             dlg.FfbTextBox_Write(String.Format("\r\n >> Direction: {0} deg ({1:D2})",\
                                    Polar2Deg(Effect.Direction), Effect.Direction));
         else
         {
           dlg.FfbTextBox_Write(String.Format("\r\n >> X Direction: {0:X2}", Effect.DirX));
           dlg.FfbTextBox_Write(String.Format("\r\n >> Y Direction: {0:X2}", Effect.DirY));
         };

         // Duration of the effect
         if (Effect.Duration == 0xFFFF)
             dlg.FfbTextBox_Write(String.Format("\r\n >> Duration: Infinit"));
         else
             dlg.FfbTextBox_Write(String.Format("\r\n >> Duration: {0} MilliSec", \
                                                (int)(Effect.Duration)));

         // Trigger Repeat
         if (Effect.TrigerRpt == 0xFFFF)
             dlg.FfbTextBox_Write(String.Format("\r\n >> Trigger Repeat: Infinit"));
         else
             dlg.FfbTextBox_Write(String.Format("\r\n >> Trigger Repeat: {0}", \
                                                (int)(Effect.TrigerRpt)));

         // Sample Period
         if (Effect.SamplePrd == 0xFFFF)
             dlg.FfbTextBox_Write(String.Format("\r\n >> Sample Period: Infinit"));
         else
             dlg.FfbTextBox_Write(String.Format("\r\n >> Sample Period: {0}", \
                                                (int)(Effect.SamplePrd)));


         // Gain
        dlg.FfbTextBox_Write(String.Format("\r\n >> Gain: {0}%",Byte2Percent(Effect.Gain)));
       };
}
```

Ffb_h_Eff_Ramp function

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

Syntax

C#

```
UInt32 Ffb_h_Eff_Ramp(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       vJoy.FFB_EFF_RAMP RampEffect = new vJoy.FFB_EFF_RAMP();
       if (0 == joystick.Ffb_h_Eff_Ramp(data, ref RampEffect))
       {
        dlg.FfbTextBox_Write(String.Format("\r\n >> Ramp Start: {0}", RampEffect.Start));
        dlg.FfbTextBox_Write(String.Format("\r\n >> Ramp End: {0}", RampEffect.End));
       };
}
```

Ffb_h_EffOp function

Extract information from **operation** FFB data packet

Syntax

C#

```
UInt32 Ffb_h_EffOp(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       vJoy.FFB_EFF_OP Operation = new vJoy.FFB_EFF_OP();
       string EffOpStr = "";
       if (0 == joystick.Ffb_h_EffOp(data, ref Operation)\
                     && EffectOpStr(Operation.EffectOp, ref EffOpStr))
       {
        dlg.FfbTextBox_Write(String.Format("\r\n >> Effect Operation: {0}", EffOpStr));
        if (Operation.LoopCount == 0xFF)
          dlg.FfbTextBox_Write(String.Format("\r\n >> Loop until stopped"));
        else
          dlg.FfbTextBox_Write(String.Format("\r\n >> Loop {0} times",\
                                                (int)(Operation.LoopCount)));
       };
}
```

Ffb_h_DevCtrl function

Extract information from device-wide control instructions FFB data packet

Syntax

C#

```
UInt32 Ffb_h_DevCtrl(
      IntPtr Packet,
      ref 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. 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 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#

```
UInt32 Ffb_h_Eff_Period(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       vJoy.FFB_EFF_PERIOD EffPrd = new vJoy.FFB_EFF_PERIOD();
       if (0 == joystick.Ffb_h_Eff_Period(data, ref EffPrd))
       {
        dlg.FfbTextBox_Write(String.Format("\r\n >> Magnitude: {0}", EffPrd.Magnitude ));
        dlg.FfbTextBox_Write(String.Format("\r\n >> Offset: {0}", EffPrd.Offset));
        dlg.FfbTextBox_Write(String.Format("\r\n >> Phase: {0}", EffPrd.Phase));
        dlg.FfbTextBox_Write(String.Format("\r\n >> Period: {0}", (int)(EffPrd.Period)));
       };

}
```

Ffb_h_Eff_Cond function

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

Syntax

C#

```
UInt32 Ffb_h_Eff_Cond(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       vJoy.FFB_EFF_COND Condition = new vJoy.FFB_EFF_COND();
       if (0 == joystick.Ffb_h_Eff_Cond(data, ref Condition))
       {
         if (Condition.isY)
           dlg.FfbTextBox_Write(String.Format("\r\n >> Y Axis"));
         else
           dlg.FfbTextBox_Write(String.Format("\r\n >> X Axis"));

           dlg.FfbTextBox_Write(String.Format("\r\n >> Center Point Offset: {0}", \
                                                Condition.CenterPointOffset));
           dlg.FfbTextBox_Write(String.Format("\r\n >> Positive Coefficient: {0}", \
                                                Condition.PosCoeff));
           dlg.FfbTextBox_Write(String.Format("\r\n >> Negative Coefficient: {0}", \
                                                Condition.NegCoeff));
           dlg.FfbTextBox_Write(String.Format("\r\n >> Positive Saturation: {0}", \
                                                Condition.PosSatur ));
           dlg.FfbTextBox_Write(String.Format("\r\n >> Negative Saturation: {0}", \
                                                Condition.NegSatur));
           dlg.FfbTextBox_Write(String.Format("\r\n >> Dead Band: {0}", \
                                                Condition.DeadBand));
       }
}
```

Ffb_h_DevGain function

Extract information from FFB data packet : Device Global gain.

Syntax

C#

```
UInt32 Ffb_h_DevGain(
      IntPtr Packet
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
       Byte Gain = 0;
       if (0 == joystick.Ffb_h_DevGain(data, ref Gain))
        dlg.FfbTextBox_Write(String.Format("\r\n >> Global Device Gain: {0}",\
                                                          Byte2Percent(Gain)));
}
```

Ffb_h_Eff_Envlp function

Extract information from FFB data packet : Effect Envelope block.

Syntax

C#

```
      UInt32 Ffb_h_Eff_Envlp(
            IntPtr Packet
            ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
 vJoy.FFB_EFF_ENVLP Envelope = new vJoy.FFB_EFF_ENVLP();
 if (0 == joystick.Ffb_h_Eff_Envlp(data, ref Envelope))
 {
   dlg.FfbTextBox_Write(String.Format("\r\n >> Attack Level: {0}", Envelope.AttackLevel));
   dlg.FfbTextBox_Write(String.Format("\r\n >> Fade Level: {0}", Envelope.FadeLevel ));
   dlg.FfbTextBox_Write(String.Format("\r\n >> Attack Time: {0}",\
                                                (int)(Envelope.AttackTime)));
   dlg.FfbTextBox_Write(String.Format("\r\n >> Fade Time: {0}", (int)(Envelope.FadeTime)));
 };
}
```

Ffb_h_EffNew function

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

Syntax

C#

```
UInt32 Ffb_h_EffNew(
      IntPtr Packet,
      ref 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#

```
UInt32 Ffb_h_Eff_Constant(
      IntPtr Packet,
      ref 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
private static void OnEffectObj(IntPtr data, object userData)
{
 vJoy.FFB_EFF_CONSTANT ConstantEffect = new vJoy.FFB_EFF_CONSTANT();
 if (0 == joystick.Ffb_h_Eff_Constant(data, ref ConstantEffect))
       dlg.FfbTextBox_Write(String.Format("\r\n >> Constant Magnitude: {0}",\
                                                    ConstantEffect.Magnitude));
}
```

Interface Structures

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

JoystickState Structure

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

Syntax

C#

```
[StructLayout(LayoutKind.Sequential)] public struct JoystickState
{
    public byte bDevice;
    public Int32 Throttle;
    public Int32 Rudder;
    public Int32 Aileron;
    public Int32 AxisX;
    public Int32 AxisY;
    public Int32 AxisZ;
    public Int32 AxisXRot;
    public Int32 AxisYRot;
    public Int32 AxisZRot;
    public Int32 Slider;
    public Int32 Dial;
    public Int32 Wheel;
    public Int32 AxisVX;
    public Int32 AxisVY;
    public Int32 AxisVZ;
    public Int32 AxisVBRX;
    public Int32 AxisVBRY;
    public Int32 AxisVBRZ;
    public UInt32 Buttons;
    public UInt32 bHats;
    public UInt32 bHatsEx1;
    public UInt32 bHatsEx2;
    public UInt32 bHatsEx3;
    public UInt32 ButtonsEx1;
    public UInt32 ButtonsEx2;
    public UInt32 ButtonsEx3;
};
```

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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_REPORT
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public FFBEType EffectType;
    [FieldOffset(8)]
    public UInt16 Duration;
    [FieldOffset(10)]
    public UInt16 TrigerRpt;
    [FieldOffset(12)]
    public UInt16 SamplePrd;
    [FieldOffset(14)]
    public Byte Gain;
    [FieldOffset(15)]
    public Byte TrigerBtn;
    [FieldOffset(16)]
    public bool Polar;
    [FieldOffset(20)]
    public Byte Direction;
    [FieldOffset(20)]
    public Byte DirX;
    [FieldOffset(21)]
    public Byte DirY;
}
```

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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_RAMP
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public Int16 Start;
    [FieldOffset(8)]
    public Int16 End;
}
```

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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_OP
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public FFBOP EffectOp;
    [FieldOffset(8)]
    public Byte LoopCount;
}
```

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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_PERIOD
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public UInt32 Magnitude;
    [FieldOffset(8)]
    public Int16 Offset;
    [FieldOffset(12)]
    public UInt32 Phase;
    [FieldOffset(16)]
    public UInt32 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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_COND
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public bool isY;
    [FieldOffset(8)]
    public Int16 CenterPointOffset;
    [FieldOffset(12)]
    public Int16 PosCoeff;
    [FieldOffset(16)]
    public Int16 NegCoeff;
    [FieldOffset(20)]
    public UInt32 PosSatur;
    [FieldOffset(24)]
    public UInt32 NegSatur;
    [FieldOffset(28)]
    public Int32 DeadBand;
}
```

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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_ENVLP
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public UInt16 AttackLevel;
    [FieldOffset(8)]
    public UInt16 FadeLevel;
    [FieldOffset(12)]
    public UInt32 AttackTime;
    [FieldOffset(16)]
    public UInt32 FadeTime;
}
```

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#

```
[StructLayout(LayoutKind.Explicit)]
public struct FFB_EFF_CONSTANT
{
    [FieldOffset(0)]
    public Byte EffectBlockIndex;
    [FieldOffset(4)]
    public Int16 Magnitude;
}
```

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     =  0x01,
	PT_ENVREP     =  0x02,
	PT_CONDREP    =  0x03,
	PT_PRIDREP    =  0x04,
	PT_CONSTREP   =  0x05,
	PT_RAMPREP    =  0x06,
	PT_CSTMREP    =  0x07,
	PT_SMPLREP    =  0x08,
	PT_EFOPREP    =  0x0A,
	PT_BLKFRREP   =  0x0B,
	PT_CTRLREP    =  0x0C,
	PT_GAINREP    =  0x0D,
	PT_SETCREP    =  0x0E,

	// Feature
	PT_NEWEFREP   =  0x01+0x10,
	PT_BLKLDREP   =  0x02+0x10,
	PT_POOLREP    =  0x03+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

FfbCbFunc function pointer

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

Syntax

C#

```
delegate void FfbCbFunc(
      IntPtr FfbPacket,
      object 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.