

C# vJoy Feeder SDK

Version 2.0.1 – October 2012

Table of Contents

C# vJoy Feeder SDK.....	1
File listing:.....	1
Fundamentals:.....	2
Reccomended Practices:.....	3
Test vJoy Driver:.....	3
Test vJoy Virtual Devices:.....	3
Acquire the vJoy Device:.....	4
Feed vJoy Device:.....	4
Relinquish the vJoy Device:.....	7
Interface Function Reference:.....	8
General driver data.....	8
Write access to vJoy Device.....	8
vJoy Device properties.....	8
Robust write access to vJoy Devices.....	10
Build & Deploy:.....	11

This SDK includes all that is needed to write a feeder for vJoy version 2.0.1

Check for the latest [SDK](#).

File listing:

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

Fundamentals:

This interface and example will enable you to write a C# vJoy feeder.

To write a C/C++ refer to ReadMe file in parent folder.

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

Test Driver:	Check that the driver is installed and enabled. Obtain information about the driver. An installed driver implies at least one vJoy device.
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 <i>owned</i> or is <i>free</i> .
Updating:	Inject <u>position data</u> 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 <i>owned</i> by the feeder and cannot be fed by another application until relinquished.

Notes:

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

Reccomended Practices:

Test vJoy Driver:

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

```
joystick = new vJoy();

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

Test vJoy Virtual Devices:

Check if device is installed and what its state:

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

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

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

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

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

Acquire the vJoy Device:

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

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

Feed vJoy Device:

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

There are two approaches:

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

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

The second approach hides the details of the data fed to the device at the expanse of excessive calls to the device driver.

Efficient:

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

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

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

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

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

}; // While
```

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

Robust:

```
// Reset this device to default values
joystick.ResetVJD(id);

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

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

    // If Continuous POV hat switches installed - make them go round
    // For high values - put the switches in neutral state
    if (ContPovNumber>0)
    {
        if ((count * 70) < 30000)
        {
            res = joystick.SetContPov(((int)count * 70), id, 1);
            res = joystick.SetContPov(((int)count * 70) + 2000, id, 2);
            res = joystick.SetContPov(((int)count * 70) + 4000, id, 3);
            res = joystick.SetContPov(((int)count * 70) + 6000, id, 4);
        }
        else
        {
            res = joystick.SetContPov(-1, id, 1);
            res = joystick.SetContPov(-1, id, 2);
            res = joystick.SetContPov(-1, id, 3);
            res = joystick.SetContPov(-1, id, 4);
        }
    };

    // If Discrete POV hat switches installed - make them go round
    // From time to time - put the switches in neutral state
    if (DiscPovNumber>0)
    {
        if (count < 550)
        {
            joystick.SetDiscPov((((int)count / 20) + 0) % 4, id, 1);
            joystick.SetDiscPov((((int)count / 20) + 1) % 4, id, 2);
            joystick.SetDiscPov((((int)count / 20) + 2) % 4, id, 3);
            joystick.SetDiscPov((((int)count / 20) + 3) % 4, id, 4);
        }
        else
        {
            joystick.SetDiscPov(-1, id, 1);
            joystick.SetDiscPov(-1, id, 2);
            joystick.SetDiscPov(-1, id, 3);
            joystick.SetDiscPov(-1, id, 4);
        }
    };

    System.Threading.Thread.Sleep(20);
} // While (Robust)
```

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

Relinquish the vJoy Device:

You must relinquish the device when the driver exits:

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

Interface Function Reference:

General driver data

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

```
bool vJoyEnabled();
```

Returns **true** if vJoy version 2.x is installed and enabled.

```
short GetvJoyVersion();
```

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

```
string GetvJoyProductString();
```

```
string GetvJoyManufacturerString();
```

```
string GetvJoySerialNumberString();
```

To be used only after `vJoyEnabled()`

Write access to vJoy Device

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

VJD stands for *Virtual Joystick Device*.

```
VjdStat GetVJDStatus(UINT32 rID);
```

Returns the status of the specified device

The status can be one of the following values:

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

```
bool AcquireVJD(UINT32 rID);
```

Acquire the specified device.

Only a device in state VJD_STAT_FREE can be acquired.

If acquisition is successful the function returns TRUE and the device status becomes VJD_STAT_OWN.

```
void RelinquishVJD(UINT32 rID);
```

Relinquish the previously acquired specified device.

Use only when device is state VJD_STAT_OWN.

State becomes VJD_STAT_FREE immediately after this function returns.

```
bool UpdateVJD(UINT32 rID, ref JoystickState pData);
```

Update the position data of the specified device.

Use only after device has been successfully acquired.

Input parameter is a reference to structure of type `JoystickState` that holds the position data.

Returns **true** if device updated.

vJoy Device properties

The following functions receive the virtual device ID (rID) and return the relevant data.

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

```
int GetVJDButtonNumber(uint rID);
```

Returns the number of buttons in the specified device.

Valid values are 0 to 32

```
int GetVJDDiscPovNumber(uint rID);
```

Returns the number of discrete-type POV hats in the specified device

Discrete-type POV Hat values may be North, East, South, West or neutral

Valid values are 0 to 4 (in version 2.0.1)

```
int GetVJDContPovNumber(uint rID);
```

Returns the number of continuous-type POV hats in the specified device

continuous-type POV Hat values may be 0 to 35900

Valid values are 0 to 4 (in version 2.0.1)

```
bool GetVJDAxisExist(UINT32 rID, HID_USAGES Axis);
```

Returns TRUE if the specified axis exists in the specified device

Axis values can be:

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

Robust write access to vJoy Devices

The following functions receive the virtual device ID (rID) and return the relevant data.

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

```
bool ResetVJD(UInt32 rID);
```

Resets all the controls of the specified device to a set of values.

These values are hard coded in the interface DLL and are currently set as follows:

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

```
bool ResetAll();
```

Resets all the controls of the all devices to a set of values.

See function Reset VJD for details.

```
bool ResetButtons(UInt32 rID);
```

Resets all buttons (To 0) in the specified device.

```
bool ResetPovs(UInt32 rID);
```

Resets all POV Switches (To -1) in the specified device.

```
bool SetAxis(Int32 Value, UInt32 rID, HID_USAGES Axis);
```

Write Value to a given axis defined in the specified VDJ.

Axis values can be:

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

```
bool SetBtn(bool Value, UInt32 rID, uint nBtn);
```

Write Value (true or false) to a given button defined in the specified VDJ.

nBtn can range 1-32

```
bool SetDiscPov(Int32 Value, UInt32 rID, uint nPov);
```

Write Value to a given discrete POV defined in the specified VDJ

Value can be one of the following:

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

nPov selects the destination POV Switch. It can be 1 to 4

```
bool SetContPov(Int32 Value, UInt32 rID, uint nPov);
```

Write Value to a given continuous POV defined in the specified VDJ

Value can be in the range: -1 to 35999. It is measured in units of one-hundredth a degree. -1 means Neutral (Nothing pressed).

nPov selects the destination POV Switch. It can be 1 to 4

Build & Deploy:

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

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