vGenInterface

vJoy and vXbox API

Table of Contents

General

vGenInterface is a collection of 3 APIs that can be used for writing a feeder for vXbox devices and for a mixture of vXbox/vJoy devices. It is also possible to use it for writing a vJoy-only feeder but it is currently recommended to use vJoyInterface API for that purpose.
vGenInterface offers three sets of interface functions:

1. Common API: Equally useful for vJoy and vXbox.
2. vJoy API: Backward Compatible with vJoyInterface. Also supports vXbox.
3. vXbox API: Tailored for vXbox feeders. Cannot be used for vJoy devices.

You can use a mixture of the above APIs if needed.

Which API

Each API has it advantages and disadvantages. In addition, you can mix them without difficulty. You should first decide which kind of feeder you are writing:

vJoy Only:

If your feeder is designed to feed only vJoy devices, you should not use this API. Use **vJoyInterface.dll** instead.

Convert a Feeder:

Your feeder is written for vJoy devices and you use API file vJoyInterface.dll.
By linking to this (vGenInterface.dll) API you should get exactly the same results when target devices are vJoy devices.
If you want to add support for vXbox devices, it is recommended that you make at least the following minimal changes in your code:

1. Convert vXbox device index values to range 1001-1004
2. Report to the user the LED number of the vXbox device.

Additional changes you might introduce to improve your feeder:

1. Use vXbox virtual bus administrative functions:
    1. Call function `isVBusExist()` to verify that the system supports vXbox.
    2. Call function `GetNumEmptyBusSlots()` to see if you can add a new vXbox device.
2. If you want to support FFB add an FFB thread and poll Vibration status by repeatedly calling `GetVibration()` and passing vibration values to your hardware.

New Feeders:

If the new feeder is designed for only one of the target types, you can safely use either vXbox API or vJoyInterface.dll.
If the new feeder is designed to support both vJoy and vXbox, you should decide whether you want to take the agnostic approach or not.
Agnostic: Most operations will treat devices regardless to their type (vJoy/vXbox).
This approach simplifies the code but does not take advantage of the specifics of each type.

Example: you might need to check if button 10 exists before using it though it always exists for vXbox device.

Non-Agnostic: Most operations will treat devices according to their type (vJoy/vXbox).

This approach requires per-type coding making the code larger. However, it will enable the code to take advantage of the particular features of each type.

Example: You might take advantage of the large number of buttons in vJoy devices.

Note: It is OK to mix the approaches in your feeder code.

vJoy API

This is an expansion of vJoyInterface API. It covers vJoy devices and vXbox devices.

The API treats **vJoy** devices exactly as vJoyInterface API.

The API treats **vXbox** devices in a similar way with a few exceptions:

- vXbox devices are identified by their ID. You will need to add 1000 (Decimal) to the original device ID hence for this API, the vXbox devices are in the range 1001-1004.
- No FFB support. Use specific vXbox FFB functions
- No LED-number support
- D-Pad: Supports only North, East, South and West.
- Axes (Thumb-stick): Lower resolution

Common API

Device Status

The following functions are used to change and to inquire the status of a virtual device.

Feeder life-cycle involve the **acquisition** of a virtual device, using it and finally **relinquishing** the device.

Only **free** devices can be acquired and by acquiring a device it becomes **owned** by the feeder. It is sometimes useful to know whether a device **exists** even if it is owned by another feeder.

Device State	**vJoy Status**	**vXbox Status**
Exists	OWNED or FREE or BUSY	Plugged-in
Owned	OWNED	Plugged-in by current process
Free	FREE	Not Plugged-in

AcquireDev function

Acquire a Virtual Device

Syntax

```
DWORD AcquireDev(UINT DevId, DevType dType, HDEVICE * hDev);
```

Parameters

DevId

[in] ID of device to acquire

dType

[in] Type of device to acquire (vJoy or vXbox)

hDev

[out] Pointer that receives the handle to the acquired device

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If parameter dType is invalid, the return code is **STATUS_INVALID_PARAMETER_2**.

If the function fails and the type of the device is **vXbox**, see function PlugIn for return values.

If the function fails and the type of the device is **vJoy**, the return code is **STATUS_UNSUCCESSFUL**.

Remarks

Use this function to plug-in a new vXbox device or to acquire a new vJoy device. If the function succeeds it returns **STATUS_SUCCESS** and sets *hDev* to point to the handle to the device.

Once the device acquired, it is **owned** by the application and seen as **existing** or plugged-in by other applications.

To remove the device call function RelinquishDev(hDev) where the *hDev* parameter is the handle acquired by function AcquireDev().

Parameter *DevId* is the ID of the device to be acquired. If the device type is **vJoy** then it must be the ID of a free device, in the rage between 1 and 16. If the device type is **vXbox** then it must be the ID of an empty slot, in the rage between 1 and 4.

RelinquishDev function

Relinquish a Virtual Device

Syntax

```
DWORD RelinquishDev(HDEVICE  hDev);
```

Parameters

hDev

[in] The handle of the device to be relinquished.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If parameter *hDev* is an invalid handle, the return code is **STATUS_INVALID_HANDLE**.

If the function fails and the type of the device is **vXbox**, see function UnPlug for return values.

If the function fails and the type of the device is **vJoy**, the return code is **STATUS_UNSUCCESSFUL**.

Remarks

Use this function to unplug an owned vXbox device or to relinquish an owned vJoy device. You can only relinquish an owned device.

Once the function has successfully returned, the handle to the device cannot be used. It is recommended that you mark it as invalid by setting the handle variable to INVALID_DEV.

isDevFree function

Test whether a Virtual Device is Free.

Syntax

```
DWORD isDevFree(UINT DevId, DevType dType, BOOL * Free);
```

Parameters

DevId

[in] ID of device to inquire.

dType

[in] Type (**vJoy** or **vXbox**) of device to inquire.

Free

[out] Result: TRUE if device is Free. FALSE otherwise.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *Free* is invalid, the function returns **STATUS_INVALID_PARAMETER_3.**

Other return codes are possible.

Remarks

Use this function to checks whether a device is Free.

A Free device is not owned by any application.

Only Free devices can be acquired.

isDevOwned function

Verify that a Virtual Device is owned.

Syntax

```
DWORD isDevOwned(UINT DevId, DevType dType, BOOL * Owned);
```

Parameters

DevId

[in] ID of device to inquire.

dType

[in] Type (**vJoy** or **vXbox**) of device to inquire.

Owned

[out] Result: TRUE if owned FALSE otherwise.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *Owned* is invalid, the function returns **STATUS_INVALID_PARAMETER_3.**

Other return codes are possible.

Remarks

Use this function to checks if a device is owned by the current application. This function is useful to verify that a device is capable of receiving position data and if it is possible to relinquish the device.

An application can Relinquish (or unplug) only an owned device.

An application can feed with position data only an owned device.

Only an owned device has a valid handle.

isDevExist function

Verify that a Virtual Device exists.

Syntax

```
DWORD isDevExist(UINT DevId, DevType dType, BOOL * Exist);
```

Parameters

DevId

[in] ID of device to inquire.

dType

[in] Type (**vJoy** or **vXbox**) of device to inquire.

Exist

[out] Result: TRUE if device exists FALSE otherwise.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *Owned* is invalid, the function returns **STATUS_INVALID_PARAMETER_3.**

Other return codes are possible.

Remarks

Use this function to checks if a device exists.

An existing device is free, owned by the current application or by another application.

A **vJoy** device that exists is a device that has already been configured.

A **vXbox** device that exists is a device that has already been plugged-in.

Device Information

The following functions are used to get information about an owned device.

Some information is related to the device capabilities such as the number of buttons, Hat switches and axes in the device.

Use the functions that inquire for the device capabilities to determine the existing controls while it is not an error to try to feed a non-existing control.

Some information is related to administrative data such as the device' type or the device' id.

Administrative data can be used if you want to mix function from all vGenInterface APIs.

You can obtain device ID and Type from a device Handle to be used by Backward Compatibility API or by the vXbox API.

You can obtain device Handle from device ID and Type, if the device was not created by the Common API.

You can also obtain the LED number of a vXbox device.

isAxisExist function

For an owned Virtual Device, check whether a specific axis exists.

Syntax

```
DWORD isAxisExist(HDEVICE hDev, UINT nAxis, BOOL * Exist);
```

Parameters

hDev

[in] Handle of device to inquire.

nAxis

[in] Axis number (See remarks).

Exist

[out] Pointer to value: TRUE is axis exists, FALSE otherwise.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *Exist* is invalid, the function returns **STATUS_INVALID_PARAMETER_3.**

If the device was removed (possibly by another application), the function returns **STATUS_DEVICE_REMOVED**.

Remarks

Use this function to check if a specific axis is configured in the specified virtual device.

May be used for owned devices only.

Useful mainly for **vJoy** devices because they are configurable. **vXbox** devices are not configurable so they will always report axes 1 to 6 as existing.

The axes are enumerated as follows:

#	vJoy	vXbox
1	X	X
2	Y	Y
3	Z	Trigger (R)
4	Rx	RX
5	Ry	RY
6	Rz	Trigger (L)
7	Slider 0	-
8	Slider 1	-

GetDevButtonN function

For an owned Virtual Device, get the number of buttons.

Syntax

```
DWORD GetDevButtonN(HDEVICE hDev, UINT * nBtn);
```

Parameters

hDev

[in] Handle of device to inquire.

nBtn

[out] Pointer to the number of buttons.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *nBtn* is invalid, the function returns **STATUS_INVALID_PARAMETER_2.**

If the device was removed (possibly by another application), the function returns **STATUS_DEVICE_REMOVED**.

Remarks

Use this function to get the number of buttons configured in the specified virtual device.

May be used for owned devices only.

Useful mainly for **vJoy** devices because they are configurable. **vXbox** devices are not configurable so they will always report 10 buttons.

GetDevHatN function

For an owned Virtual Device, get the number of Hat Switches (a.k.a. POV) or D-Pads.

Syntax

```
DWORD DWORD GetDevHatN(HDEVICE hDev, UINT * nHat);
```

Parameters

hDev

[in] Handle of device to inquire.

nHat

[out] Pointer to the number of buttons.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *nHat* is invalid, the function returns **STATUS_INVALID_PARAMETER_2.**

If the device was removed (possibly by another application), the function returns **STATUS_DEVICE_REMOVED**.

Remarks

Use this function to get the number of Hat switches or D-Pads configured in the specified virtual device.

May be used for owned devices only.

Useful mainly for **vJoy** devices because they are configurable. **vXbox** devices are not configurable so they will always report 1 D-Pad.

GetDevHandle function

Retrieve the device handle from the devices' container.

Syntax

```
DWORD GetDevHandle(UINT DevId, DevType dType, HDEVICE * hDev);
```

Parameters

DevId

[in] ID of device to inquire.

dType

[in] Type (**vJoy** or **vXbox**) of device to inquire.

hDev

[out] Pointer to the handle corresponding to the indicated device.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle belongs to a device that is no longer owned, the return code is **STATUS_DEVICE_REMOVED**.

If pointer *hDev* is invalid, the function returns **STATUS_INVALID_PARAMETER_3.**

Other return codes are possible.

Remarks

This function is needed in cases where a device was Plugged-in or Acquired by another part of the API that does not return a device handle.

The function searches the device-container for a device that corresponds to the given ID/Type pair. If it finds it, the function verifies that the device is owned by the calling application and returns the handle if verified.

If a handle is found but it no longer represents an owned device, the function removes the entry from the device-container and returns **STATUS_DEVICE_REMOVED**.

GetDevId function

Get the ID of an owned Virtual Device.

Syntax

```
DWORD GetDevId(HDEVICE hDev, UINT * dID);
```

Parameters

hDev

[in] Handle of device to inquire.

dID

[out] ID of a device.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *dID* is invalid, the function returns **STATUS_INVALID_PARAMETER_2.**

If the device was removed (possibly by another application), the function returns **STATUS_DEVICE_REMOVED**.

Remarks

Use this function to retrieve the ID of an owned **vXbox** or **vJoy** device from its handle.

If the function is successful, *dID* will point to the ID of the virtual device.

Virtual Xbox devices (vXbox) have bus-wide unique **ID**s (Range 1-4). Do not confuse between vXbox IDs and Xbox LED Numbers.

GetDevType function

Get the type of an owned Virtual Device.

Syntax

```
DWORD GetDevType(HDEVICE hDev, DevType * dType);
```

Parameters

hDev

[in] Handle of device to inquire.

dType

[out] Type of device (**vJoy** or **vXbox**).

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *pType* is invalid, the function returns **STATUS_INVALID_PARAMETER_2.**

If the device was removed (possibly by another application), the function returns **STATUS_DEVICE_REMOVED**.

Remarks

Use this function to retrieve the type of an owned virtual device from its handle.

If the function is successful, *dType* will point to one of the predefined types (vJoy or vXbox).

GetDevNumber function

Get the ID or LED Number of an owned Virtual Device.

Syntax

```
DWORD GetDevNumber(HDEVICE hDev, UINT * dNumber);
```

Parameters

hDev

[in] Handle of device to inquire.

dNumber

[out] LED Number of a **vXbox** device or the ID of a **vJoy** device.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If pointer *pNumber* is invalid, the function returns **STATUS_INVALID_PARAMETER_2.**

If the device was removed (possibly by another application), the function returns **STATUS_DEVICE_REMOVED**.

Remarks

Use this function to retrieve the LED Number of an owned **vXbox** device from its handle. For **vJoy** device the function retrieves the device ID.

If the function is successful, *dNumber* will point to the LED Number of the vXbox device or to the ID of the vJoy device.

Every plugged-in Xbox device (Physical or virtual) receives from the system a unique **LED number**. Physical devices Illuminate a quarter of a circle LED that indicates it's number. In addition virtual Xbox devices (vXbox), have bus-wide unique **ID**s (Range 1-4). Do not confuse between vXbox IDs and Xbox LED Numbers.

Device Feeding

The following functions are used to feed the virtual device with position data. Position data is the exact position of each vJoy axes, vXbox thumb-stick, vXbox trigger, button and Hat Switch.

Hat switches as sometimes referred to as POVs. The vXbox equivalent is D-Pad.

The axes, thumb-stick and triggers have specific names. However, the common API enumerates them as follows:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
vJoy	X	Y	Z	Rx	Ry	Rz	Slider 0	Slider 1
vXbox	Left-X	Left-Y	Right-Trigger	Right-X	Right-Y	Left-Trigger	-	-

The axes, thumb-stick and triggers have different ranges. However, the common API feeds them with position values of 0.00 to 100.00. The actual positions are mapped as follows:

Value	**vJoy Axes**	**vXbox**	
		Axes	**Triggers**
0	0	-32768	0
50	16384	0	127
100	32768	32767	255

A **vJoy** device may contain up to four Hat switch devices enumerated 1 to 4. The Hat switches may be **continuous** or **discrete**. Continuous Hat switches position ranges from 0° to 359.99°. Discrete Hat switches may be in one of 4 positions: 0° (North), 90° (East), 180° (South), 270° (West). Hat switches (of both types) may also be in Neutral position.

A **vXbox** device has one D-Pad which is referred to as Hat switch number 1. It may be in the Neutral position or in one of the eight positions as described in the following table:

Value	Position		Applies to:		
			Continuous (vJoy)	Discrete (vJoy)	D-Pad
0%	North	0°	✓	✓	✓
12.5%	North-East	45°	✓		✓
25%	East	90°	✓	✓	✓
37.5%	South-East	135°	✓		✓
50%	South	180°	✓	✓	✓
62.5%	South-West	225°	✓		✓
75%	West	270°	✓	✓	✓
87.5%	North-West	315°	✓		✓
99%	Almost North	356.4°	✓		

Note the the common API feeds the Hat with position values of 0.00 to 99.99.

A **vJoy** device may contain up to 128 buttons. However, the Common API supports only the first 32 buttons. The buttons are simply enumerated 1 to 32.

A **vXbox** device always contains ten buttons. Each button has a unique function. However, the common API refers to them by a serial number as follows:

1	2	3	4	5	6	7	8	9	10
A	B	X	Y	Left-Shoulder	Right-Shoulder	Back	Start	Left-thumb	Right-thumb

SetDevAxis function

Set position of virtual device axis or trigger.

Syntax

```
DWORD SetDevAxis(HDEVICE hDev, UINT Axis, FLOAT Value);
```

Parameters

hDev

[in] Handle of device to feed.

Axis

[in] Axis or trigger to change.

Value

[in] Position of axis in the range 0 to 100

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set an axis or trigger control of the specified device to the required position.

Meaning of *Axis* parameter:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
vJoy	X	Y	Z	Rx	Ry	Rz	Slider 0	Slider 1
vXbox	Left-X	Left-Y	Right-Trigger	Right-X	Right-Y	Left-Trigger	-	-

Parameter *Value* ranges between 0 and 100 and is passed as a FLOAT to enable high precision. The value is mapped to an internal value based on the specifics of the target axis:

Value	vJoy Axes	vXbox	
		Axes	Triggers
0	0	-32768	0
50	16384	0	127
100	32768	32767	255

SetDevPov function

Set position of virtual Pov (Hat Switch) or D-Pad.

Syntax

```
DWORD SetDevPov(HDEVICE hDev, UINT nPov, FLOAT Value);
```

Parameters

hDev

[in] Handle of device to feed.

nPov

[in] Hat switch (a.k.a. POV) or D-Pad to control

Value

[in] Position of switch in degrees (range 0 to 359.99)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set a POV Hat switch or D-Pad control of the specified device to the required position.

vJoy supports up to four POV hat switches that can be of two types: Continuous or Discrete.

vXbox supports exactly one D-Pad which supports eight directions.

These controls can also be in neutral state.

POV hat Switches are accessed by their serial number (1-4). The D-Pad is considered as POV hat Switch number 1.

The value passed to a POV hat Switch/D-pad control ranges between 0 and 359.99. Value -1 corresponds to the neutral state.

Example Values:

Value	Position		Applies to:		
			Continuous (vJoy)	Discrete (vJoy)	D-Pad
0.00	North	0°	✓	✓	✓
45.00	North-East	45°	✓		✓
90.00	East	90°	✓	✓	✓
135.00	South-East	135°	✓		✓
180.00	South	180°	✓	✓	✓
225.00	South-West	225°	✓		✓
270.00	West	270°	✓	✓	✓
315.00	North-West	315°	✓		✓
356.33	Almost North	356.33°	✓		
-1	Neutral	-	✓	✓	✓

SetDevButton function

Press/Release one or more virtual device buttons.

Syntax

```
DWORD SetDevButton(HDEVICE hDev, UINT Button, BOOL Press);
```

Parameters

hDev

[in] Handle of device to feed.

Button

[in] Button(s) to Press or Release

Press

[in] Press or Release button

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If the function fails because the handle is invalid or if the handle belongs to a device that is no longer owned, the return code is **STATUS_INVALID_HANDLE**.

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to Press or Release one or more buttons of the specified device.

If you want to **Press** the buttons, set parameter *Press* to **TRUE**.

If you want to **Release** the buttons, set parameter *Press* to **FALSE**.

Specify the button to Press or Release by setting the correct value in parameter *Button*.

The vXbox buttons are enumerated as followed:

1	2	3	4	5	6	7	8	9	10
A	B	X	Y	Left-Shoulder	Right-Shoulder	Back	Start	Left-thumb	Right-thumb

vXbox API

Virtual vXbox bus information

The following functions return information about the Virtual USB bus that the xXbox devices may be connected to. This but presents 4 virtual slots to which vXbox devices may be plugged-in.

An Empty Slot is a slot into which it is possible to plug-in a new vXbox device.

isVBusExist function

Tests that Virtual USB bus exists.

Syntax

```
DWORD isVBusExist(void);
```

Parameters

None.

Return Value

If the function finds a valid virtual bus installed on the computer, the return code is **STATUS_SUCCESS**.

If there is no such bus, the function returns **STATUS_NO_SUCH_DEVICE.**

Remarks

Always use this function before any other vXbox related operation.

If the Virtual USB bus is absent then there's no point to continue.

GetNumEmptySlots function

Tests that Virtual USB bus exists.

Syntax

```
DWORD GetNumEmptySlots(UCHAR * nSlots);
```

Parameters

nSlots

[out] Pointer that receives the number of empty vXbox slots. Use this number to determine how many vXbox devices can still be plugged-in.

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

Remarks

Use this function if you want to test how many slots are available.

The number of available slots is the number of possible slots (4) minus the number of vXbox devices plugged-in by all feeders.

Device Status

The following functions are used to change and to inquire the status of a vXbox device.

An application may **plug-in** a vXbox device into an empty slot (or free slot). Once the devices is plugged in, the slot is no longer empty and this device is considered as **plugged-in**.

The plugged-in device is said to be **owned** by the application that plugged it in.

The virtual devices are enumerated between 1 to 4. The application can access every device (or slot) by its UserIndex. However, the UserIndex of a device is not recognized by the system. The system recognizes the device by its **LED number**.

When the application no longer needs an owned device, it should **unplug** it. By this the corresponding bus-slot becomes empty.

Only an application that owns a vXbox device can unplug it. In the rare cases that an application has crashed and its devices were not unplugged you may use **force** unplug. Use this option with extreme caution.

PlugIn function

Plugged-in a vXbox controller in the specified slot.

Syntax

```
DWORD PlugIn(UINT UserIndex);
```

Parameters

UserIndex

[in] Index of the target slot.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the virtual USB bus is missing, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If device already plugged-in in the specified socket, the function returns **STATUS_DEVICE_ALREADY_ATTACHED.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device failed to start after it has been inserted, the function will unplug it and return **STATUS_DEVICE_NOT_READY**.

If the DLL fails to create a handle for this device, the function will unplug the device and return **STATUS_INVALID_HANDLE**.

Remarks

Use this function to plug-in a vBox device in the user-specifies slot. Indices are bus-wide unique and range between 1 and 4.

Do not confuse *UserIndex* (which is a virtual bus internal index) with the system-wide recognized **LED** number (or index).

If a device is plugged-in (or exists) in a specified slot the function will fail.

After the function returns successfully, the device is Plugged-in, Owned by the current feeder and its handle is valid and may be obtained by calling function **GetDevHandle**.

To remove the device call function **UnPlug** or **UnPlugForce**.

PlugInNext function

Plugged-in a vXbox controller in the next free slot.

Syntax

```
DWORD PlugInNext(UINT * UserIndex);
```

Parameters

UserIndex

[out] Pointer that receives the device User Index.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the virtual USB bus is missing, the function returns **STATUS_NO_SUCH_DEVICE.**

If device cannot be plugged-in because all sockets already in use, the function returns **STATUS_DEVICE_ALREADY_ATTACHED.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device failed to start after it has been inserted, the function will unplug it and return **STATUS_DEVICE_NOT_READY**.

If the DLL fails to create a handle for this device, the function will unplug the device and return **STATUS_INVALID_HANDLE**.

Remarks

Use this function to plug-in a vBox device in any available slot. If successful, the selected slot index is pointed by *UserIndex*. Indices are bus-wide unique and range between 1 and 4.

Do not confuse *UserIndex* (which is a virtual bus internal index) with the system-wide recognized **LED** number (or index).

If there are no available slots (4 devices are already plugged-in) the function will fail.

After the function returns successfully, the device is Plugged-in, Owned by the current feeder and its handle is valid and may be obtained by calling function **GetDevHandle**.

To remove the device call function **UnPlug** or **UnPlugForce**

UnPlug function

Unplug an owned controller from the specified slot.

Syntax

```
DWORD UnPlug(UINT UserIndex);
```

Parameters

UserIndex

[in] Index of the device to unplug

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the virtual USB bus is missing, the function returns **STATUS_NO_SUCH_DEVICE.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER**.

If specified device does not exist, the function returns **STATUS_DEVICE_DOES_NOT_EXIST**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If Device failed to unload after a long wait, the function returns **STATUS_TIMEOUT**.

Remarks

Use this function unplug (Remove) devices from the specified slot. Indices are bus-wide unique and range between 1 and 4.

Do not confuse *UserIndex* (which is a virtual bus internal index) with the system-wide recognized **LED** number (or index).

The feeder can unplug only devices that it **owns**. The feeder owns only devices it has plugged-in. It does not own devices plugged-in by another feeder.

Only **plugged-in** device can be unplugged.

UnPlugForce function

Unplug a controller from the specified slot even if not owned.

Syntax

```
DWORD UnPlugForce(UINT UserIndex);
```

Parameters

UserIndex

[in] Index of the device to unplug

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the virtual USB bus is missing, the function returns **STATUS_NO_SUCH_DEVICE.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER**.

If specified device does not exist, the function returns **STATUS_DEVICE_DOES_NOT_EXIST**.

If Device failed to unload after a long wait, the function returns **STATUS_TIMEOUT**.

Remarks

Warning: This function will remove any device including devices that are owned by **other feeders**.

Use this function unplug (Remove) devices from the specified slot. Indices are bus-wide unique and range between 1 and 4.

Do not confuse *UserIndex* (which is a virtual bus internal index) with the system-wide recognized **LED** number (or index).

Only **plugged-in** device can be unplugged.

isControllerPluggedIn function

Tests that a vXbox controller is plugged-in in the specified slot.

Syntax

```
DWORD isControllerPluggedIn(UINT UserIndex, PBOOL Exist);
```

Parameters

UserIndex

[in] Index of the slot to test.

Exist

[out] Pointer that receives the Boolean value. Valid only when function returns **STATUS_SUCCESS**.

TRUE if a vBox device is plugged-in in the specifies slot.

FALSE if specifies slot is empty.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the virtual USB bus is missing, the function returns **STATUS_NO_SUCH_DEVICE.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If *UserIndex* is out of range , the function returns **STATUS_INVALID_PARAMETER.**

Remarks

Use this function to test existence of a vBox device in the user-specifies slot. Indices are bus-wide unique and range between 1 and 4.

Do not confuse *UserIndex* (which is a virtual bus internal index) with the system-wide recognized **LED** number (or index).

If a device is plugged-in (or exists) in a specified slot it means that another device cannot be plugged in into this slot.

The feeder can control only devices that it **owns**. The feeder owns only devices it has plugged-in. It does not own devices plugged-in by another feeder.

isControllerOwned function

Tests that a vXbox controller is plugged-in in the specified slot.

Syntax

```
DWORD isControllerOwned(UINT UserIndex, PBOOL Owned);
```

Parameters

UserIndex

[in] Index of the slot to test.

Owned

[out] Pointer that receives the Boolean value. Valid only when function returns **STATUS_SUCCESS**.

TRUE if a vBox device in the specifies slot is owned by the current feeder.

FALSE if a vBox device in the specifies slot is *not* owned by the current feeder.

Return Value

If the function successful, the return code is **STATUS_SUCCESS**.

If the virtual USB bus is missing, the function returns **STATUS_NO_SUCH_DEVICE.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If *UserIndex* is out of range , the function returns **STATUS_INVALID_PARAMETER.**

Remarks

Use this function to test if the vBox device in the user-specifies slot is owned by the current feeder. Indices are bus-wide unique and range between 1 and 4.

Do not confuse *UserIndex* (which is a virtual bus internal index) with the system-wide recognized **LED** number (or index).

The feeder can control only devices that it **owns**. The feeder owns only devices it has plugged-in. It does not own devices plugged-in by another feeder.

Only **plugged-in** device can be owned.

GetLedNumber function

Get the LED Number associated with a vXbox device..

Syntax

```
DWORD GetLedNumber(UINT UserIndex, PBYTE pLed);
```

Parameters

UserIndex

[in] Index of the device

pLed

[out] Pointer that receives the LED Number associated with the current device.

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If the DLL fails to create a handle for this device, the function will unplug the device and return **STATUS_INVALID_HANDLE**.

If pointer ***pLed*** is invalid, the function returns **STATUS_INVALID_PARAMETER_2.**

Remarks

Use this function to retrieve the LED number (a.k.a. LED Index) associated with the current device. This number represents the illuminated quarter-circle that indicates the Controller number.

The LED Number is not related to the User Index of the vXbox device.

Device Position Control

The following functions change the position of the vXbox device controls.

They change the position of the Axes, Trigger buttons, buttons and D-Pad.

The functions that control the Axes and the Trigger buttons set each control individually. The functions that control the buttons and the D-Pad may set/reset a number of controls simultaneously.

There are also functions that reset a device to its neutral position:

- Axes are centered.
- Trigger buttons not pressed
- Buttons released
- D-Pad in neutral position

SetAxisLx function

Set Left Stick X-Axis to specified value

Syntax

```
DWORD SetAxisLx(UINT UserIndex, SHORT Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Axis value (Range: -32768 to 32767)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If *Value* is out of range, the function returns **STATUS_INVALID_PARAMETER_2.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set the position of the X-Axis in the Left-thumb stick of the specified device.

The valid range is -32768 to 32767. Center value is 0.

SetAxisLy function

Set Left Stick Y-Axis to specified value

Syntax

```
DWORD SetAxisLy(UINT UserIndex, SHORT Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Axis value (Range: -32768 to 32767)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If *Value* is out of range, the function returns **STATUS_INVALID_PARAMETER_2.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set the position of the Y-Axis in the Left-thumb stick of the specified device.

The valid range is -32768 to 32767. Center value is 0.

SetAxisRx function

Set Right Stick X-Axis to specified value

Syntax

```
DWORD SetAxisRx(UINT UserIndex, SHORT Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Axis value (Range: -32768 to 32767)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If *Value* is out of range, the function returns **STATUS_INVALID_PARAMETER_2.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set the position of the X-Axis in the Right-thumb stick of the specified device.

The valid range is -32768 to 32767. Center value is 0.

SetAxisRy function

Set Right Stick Y-Axis to specified value

Syntax

```
DWORD SetAxisRy(UINT UserIndex, SHORT Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Axis value (Range: -32768 to 32767)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If *Value* is out of range, the function returns **STATUS_INVALID_PARAMETER_2.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set the position of the Y-Axis in the Right-thumb stick of the specified device.

The valid range is -32768 to 32767. Center value is 0.

SetTriggerL function

Set Left Trigger to specified value

Syntax

```
DWORD SetTriggerL(UINT UserIndex, BYTE Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Trigger value (Range: 0-255)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If *Value* is out of range, the function returns **STATUS_INVALID_PARAMETER_2.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set the position of the Left Trigger of the specified device.

The valid range is 0 to 255.

SetTriggerR function

Set Right Trigger to specified value

Syntax

```
DWORD SetTriggerR(UINT UserIndex, BYTE Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Trigger value (Range: 0-255)

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If *Value* is out of range, the function returns **STATUS_INVALID_PARAMETER_2.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to set the position of the Right Trigger of the specified device.

The valid range is 0 to 255.

SetButton function

Press/Release one or more vBox device buttons.

Syntax

```
DWORD SetButton(UINT UserIndex, WORD Button, BOOL Press);
```

Parameters

UserIndex

[in] Index of the device

Button

[in] Button(s) to Press or Release

Press

[in] Press or Release button

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to Press or Release one or more buttons of the specified device.

If you want to Press the buttons, set parameter *Press* to TRUE.

If you want to Release the buttons, set parameter *Press* to FALSE.

Specify the button(s) to Press or Release by setting the correct value in parameter *Button*. You may OR two or more button values.

Button values are defined in header file Xinput.h (part of the WDK). The available values are:

XINPUT_GAMEPAD_START	0x0010
XINPUT_GAMEPAD_BACK	0x0020
XINPUT_GAMEPAD_LEFT_THUMB	0x0040

XINPUT_GAMEPAD_RIGHT_THUMB	0x0080
XINPUT_GAMEPAD_LEFT_SHOULDER	0x0100
XINPUT_GAMEPAD_RIGHT_SHOULDER	0x0200
XINPUT_GAMEPAD_A	0x1000
XINPUT_GAMEPAD_B	0x2000
XINPUT_GAMEPAD_X	0x4000
XINPUT_GAMEPAD_Y	0x8000

SetDpad function

Press one or more vBox device D-Pad Directions.

Syntax

```
DWORD SetDpad(UINT UserIndex, UCHAR Value);
```

Parameters

UserIndex

[in] Index of the device

Value

[in] Direction(s) to press

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to Press one or more D-Pad directions of the specified device.

Specify the direction(s) to Press by setting the correct value in parameter *Value*. You may OR two or more direction values.

Every call to this function replaces the previous D-Pad state.

Use value=0 to depress all directions.

Direction values are defined in header file Xinput.h (part of the WDK). The available values are:

XINPUT_GAMEPAD_DPAD_UP	0x0001
XINPUT_GAMEPAD_DPAD_DOWN	0x0002
XINPUT_GAMEPAD_DPAD_LEFT	0x0004
XINPUT_GAMEPAD_DPAD_RIGHT	0x0008

ResetController function

Reset all vBox device controls.

Syntax

```
DWORD ResetController(UINT UserIndex);
```

Parameters

UserIndex

[in] Index of the device

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to reset all controls of the specified device.

If successful, the device controls are reset to the following values:

- Axes: Centered (Value=0)
- Triggers: Minimum (Value=0)
- Buttons: Released
- D-Pad: Neutral (centered)

ResetAllControllers function

Reset controls on all all vBox devices.

Syntax

```
DWORD ResetAllControllers(void);
```

Parameters

None

Return Value

If the function succeeds, the return code is **STATUS_SUCCESS**.

If there is no virtual USB bus, the function returns **STATUS_NO_SUCH_DEVICE.**

If *UserIndex* is out of range, the function returns **STATUS_INVALID_PARAMETER.**

If the API fails to get the information from the virtual USB bus, the function returns **STATUS_IO_DEVICE_ERROR.**

If device does not exist, the function will return **STATUS_DEVICE_DOES_NOT_EXIST**.

If device failed to start after it has been inserted, the function will return **STATUS_DEVICE_NOT_READY**.

If specified device is not owned by the current feeder, the function returns **STATUS_RESOURCE_NOT_OWNED**.

If the current position pf the device cannot be obtained, the function returns **STATUS_MEMORY_NOT_ALLOCATED**.

Remarks

Use this function to reset all controls of all vXbox device.

If successful, the device controls are reset to the following values:

- Axes: Centered (Value=0)
- Triggers: Minimum (Value=0)
- Buttons: Released
- D-Pad: Neutral (centered)