DirectInput

This section provides information about the Microsoft® DirectInput® component of the Microsoft DirectX® application programming interface (API). The DirectInput API is used to process data from a keyboard, mouse, joystick, or other game controller.

For an overview of the documentation, see Roadmap.

Roadmap

Information on using Microsoft® DirectInput® is presented in the following sections. Use this page to guide you through the documentation based on your level of experience as a DirectInput developer.

What's New in DirectInput. Summary of features new to this version of Microsoft® DirectX®. Read this section if you've used Microsoft DirectInput® before but want to take advantage of the new capabilities.

Introduction to DirectInput. High-level overview of the capabilities of DirectInput and the steps you'll take to implement it. Start with this section if you're new to DirectInput.

[C++]
Understanding DirectInput. Introduction to the objects and interfaces of DirectInput and how they are related to the Microsoft Windows® message system. Read this section if you're new to DirectInput.

[Visual Basic]
Understanding DirectInput. Introduction to the objects and classes of DirectInput and how they are related to the Microsoft Windows® message system. Read this section if you're new to DirectInput.

[C++]
Using DirectInput. Guide to the application programming interface (API). This section contains more detailed conceptual information than the previous two sections, as well as information about the API elements you'll use for setting up devices, retrieving data, and implementing force-feedback effects. If you're a novice, at least familiarize yourself with the contents of this section. Later, you can refer to topics as you need them. Use this section in conjunction with the DirectInput C/C++ Reference.

[Visual Basic]
Using DirectInput. Guide to the application programming interface (API). This section contains more detailed conceptual information than the previous two sections, as well as information about the API elements you'll use for setting up devices, retrieving data, and implementing force-feedback effects. If you're a novice, at least familiarize yourself with the contents of this section. Later, you can refer to topics as you need them. Use this section in conjunction with the DirectInput Visual Basic Reference.

Programming Tips and Tools. Advanced programming topics and a guide to the DirectInput tools provided with the DirectX software development kit (SDK).

[C++]
DirectInput C/C++ Tutorials. Step-by-step tutorials, often based on samples in the SDK. If you learn best from sample code, this is a good place to start after reading Introduction to DirectInput and Understanding DirectInput.

[Visual Basic]
DirectInput Visual Basic Tutorials. Step-by-step tutorials, often based on samples in the SDK. If you learn best from sample code, this is a good place to start after reading Introduction to DirectInput and Understanding DirectInput.

[C++]
DirectInput C/C++ Samples. A guide to the DirectInput C/C++ samples included in the SDK, illustrating the use and capabilities of DirectInput.

[Visual Basic]
DirectInput Visual Basic Samples. A guide to the DirectInput Visual Basic samples included in the SDK, illustrating the use and capabilities of DirectInput.

[C++]
DirectInput C/C++ Reference

Reference information for the API. Use this section when you need to know the details of an interface, method, structure, macro, or other element.

[Visual Basic]

DirectInput Visual Basic Reference. Reference information for the API. Use this section when you need to know the details of a class, method, type, macro, or other element.

What's New in DirectInput

The following are some of the new features in Microsoft® DirectInput®.

[C++]

Action mapping

Microsoft DirectInput for Microsoft DirectX® 8.0 introduces a major new feature: action mapping. Action mapping enables you to establish a connection between input actions and input devices that does not depend on the existence of particular device objects (such as specific buttons or axes). Action mapping simplifies the input loop and reduces the need for custom game drivers, custom device profilers, and custom configuration user interfaces in games.

For more information, see Action Mapping.

New DirectInput object features

The DirectInput object is now represented by the **IDirectInput8** interface. A new helper function, **DirectInput8Create**, creates the object and retrieves this interface. **IDirectInput8** has a new CLSID and cannot be obtained by calling **QueryInterface** on an interface to objects of the class **CLSID_DirectInput** used in earlier DirectX versions.

New keyboard properties

Two keyboard properties have been added: DIPROP_KEYNAME, which retrieves a localized key name, and DIPROP_SCANCODE, which retrieves the scan code.

Joystick slider data in rglSlider **array**

Joystick slider data which was assigned to the Z axis of a DIJOYSTATE or DIJOYSTATE2 structure under earlier DirectX versions will now be found in the **rglSlider** array of those same structures.

[Visual Basic]

Action mapping

Microsoft DirectInput for Microsoft DirectX® 8.0 introduces a major new feature: action mapping. Action mapping enables you to establish a connection between input actions and input devices that does not depend on the existence of particular device objects (such as specific buttons or axes). Action mapping simplifies the input loop and reduces the need for custom game drivers, custom device profilers, and custom configuration user interfaces in games.

For more information, see Action Mapping.

New DirectInput object features
The DirectInput object is now represented by the **DirectInput8** class, which is obtained by using the **DirectX8.DirectInputCreate** method.

New keyboard properties
Two keyboard properties have been added: DIPROP_KEYNAME, which retrieves a localized key name, and DIPROP_SCANCODE, which retrieves the scan code.

Joystick slider data in rglSlider **array**
Joystick slider data which was assigned to the Z axis of a DIJOYSTATE or DIJOYSTATE2 type under earlier DirectX versions will now be found in the **slider** array of those same types.

Introduction to DirectInput

Microsoft® DirectInput® is an API for input devices including the mouse, keyboard, joystick, and other game controllers, as well as for force-feedback (input/output) devices.

This section provides a brief overview of the capabilities of DirectInput and how to set it up in an application. Information is contained in the following topics.

- The Power of DirectInput
- Getting Started with DirectInput

For a more comprehensive view of how DirectInput works, see Understanding DirectInput.

[C++]
For a step-by-step guide to using the DirectInput application programming interface (API), see the following topics.

- Using DirectInput
- DirectInput C/C++ Tutorials

[Visual Basic]
For a step-by-step guide to using the DirectInput application programming interface (API), see the following topics.

- Using DirectInput
- DirectInput Visual Basic Tutorials

The Power of DirectInput

Apart from providing services for devices not supported by the Microsoft® Win32® API, Microsoft DirectInput® gives faster access to input data by communicating directly with the hardware drivers rather than relying on Microsoft Windows® messages.

DirectInput enables an application to retrieve data from input devices even when the application is in the background. It also provides full support for any type of input device, as well as for force feedback.

Through action mapping, applications can retrieve input data without needing to know what kind of device is being used to generate it.

The extended services and improved performance of DirectInput make it a valuable tool for games, simulations, and other real-time interactive applications running under Windows.

DirectInput does not provide any advantages for applications that use the keyboard for text entry or the mouse for navigation. For more information, see Interaction with Windows.

Getting Started with DirectInput

This topic is an overview of what is involved in setting up and using Microsoft® DirectInput® in a simple application.

[C++]
For details and examples of these steps, see Using DirectInput and DirectInput C/C++ Tutorials.

[Visual Basic]
For details and examples of these steps, see Using DirectInput and DirectInput Visual Basic Tutorials.

Note

To understand DirectInput, it is essential to understand the following terms.

- DirectInput object: The root DirectInput interface.
- Device: A keyboard, mouse, joystick, or other input device.
- DirectInputDevice object: Code representing a keyboard, mouse, joystick, or other input device.
- Device object: Code representing a key, button, trigger, and so on found on a DirectInput device object. Also called *device object instance*.

The following steps represent a simple implementation of DirectInput in which the application takes responsibility for ascertaining what device object (button, axis, and so on) generated each item of data.

1. **Create the DirectInput object.** You use methods of this object to enumerate devices and create DirectInput device objects.
2. **Enumerate devices.** This is not an essential step if you intend to use only the system mouse or keyboard. To ascertain what other input devices are available on the user's system, have DirectInput enumerate them. Each time DirectInput finds a device that matches the criteria you set, it gives you the opportunity to examine the device's capabilities. It also retrieves a unique identifier that you can use to create a DirectInput device object representing the device.
3. **Create a DirectInputDevice object for each device you want to use.** To do this, you need the unique identifier retrieved during enumeration. For the system mouse or keyboard, you can use a standard GUID.
4. **Set up the device.** For each device, first set the cooperative level, which determines the way the device is shared with other applications or the system. You must also set the data format used for identifying device objects, such as buttons and axes, within data packets. If you intend to retrieve buffered data—that is, events rather than states—you also need to set a buffer size. Optionally, at this stage you can retrieve information about the device and tailor the application's behavior accordingly. You can also set properties such as the range of values returned by joystick axes.
5. **Acquire the device.** At this stage you tell DirectInput that you are ready to receive data from the device.
6. **Retrieve data.** At regular intervals, typically on each pass through the message loop or rendering loop, get either the current state of each device or a record of events that have taken place since the last retrieval. If you prefer, you can have DirectInput notify you whenever an event occurs.
7. **Act on the data.** The application can respond either to the state of buttons and axes or to events such as a key being pressed or released.
8. **Close DirectInput.** Before exiting, your application should unacquire all devices and release them, then release the DirectInput object.

This is not the only approach to implementing DirectInput. To take advantage of the great variety of input devices available now and in the future, and to simplify configuration for the user, you can use action mapping.

By setting an action map for a device, you let DirectInput decide which device object to use for each application action. For example, instead of specifying that the throttle in your racing game must always be controlled by the y-axis, you can create an action called AXIS_THROTTLE and have DirectInput assign that action to the most appropriate axis available on the device. When you retrieve events, you identify them by their associated actions rather than by the device objects that generated them.

For more information, see Action Mapping.

Understanding DirectInput

This section covers the underlying structure of Microsoft® DirectInput® and its relationship to the Microsoft Windows® message system.

- DirectInput Objects
- Interaction with Windows

For practical information on how to implement the elements of DirectInput introduced here, see Using DirectInput.

DirectInput Objects

[C++]
An input-only Microsoft® DirectInput® implementation consists of the DirectInput object, which supports the **IDirectInput8** COM interface, and a DirectInputDevice object for each input device that provides data. Each DirectInputDevice object in turn has *device objects*, which are individual controls or switches such as keys, buttons, or axes. Device objects are also called *device object instances*.

Each DirectInputDevice object represents one input device, such as a mouse, keyboard, or joystick. In the DirectInput API, the word *joystick* means any type of input device other than a mouse or keyboard. A piece of hardware that is really a combination of different types of input devices, such as a keyboard with a touchpad, can be represented by two or more DirectInputDevice objects. A force-feedback device is represented by a single joystick object that handles both input and output.

DirectInputDevice objects instantiate the **IDirectInputDevice8** interface. The application ascertains the number and type of device objects available by using the **IDirectInputDevice8::EnumObjects** method. Individual device objects are not encapsulated as code objects, but are described in **DIDEVICEOBJECTINSTANCE** structures.

Force-feedback effects are represented by the **IDirectInputEffect** interface. Methods of this interface are used to create, modify, start, and stop effects.

All DirectInput interfaces are available in ANSI and Unicode versions. If "UNICODE" is defined during compilation, the Unicode versions are used.

[Visual Basic]
An input-only Microsoft® DirectInput® implementation consists of a single **DirectInput8** object along with **DirectInputDevice8** objects for each input device that is being used by the application. The DirectInputDevice8 object is used for retrieving the input data.

Any available device, regardless of whether it is being used by DirectInput, can also be represented by a **DirectInputDeviceInstance8** object, which can be used to retrieve miscellaneous information about that device.

Each DirectInputDevice8 object represents one input device, such as a mouse, keyboard, or joystick. (The DirectInput API uses the term *joystick* for all game controllers other than a mouse or a keyboard.) A piece of hardware that is really a combination of different types of input devices, such as a keyboard with a touchpad, can be represented by two or more DirectInputDevice8 objects. A force-feedback device is represented by a single joystick object that handles both input and output.

Each device in turn has *device objects*, which are individual controls or switches, such as keys, buttons, or axes. Each device object is represented by an instance of the **DirectInputDeviceObjectInstance** class, whose methods can be used to retrieve information about the device object. (The input data, however, is always retrieved by **DirectInputDevice8.**)

Devices and device objects can be enumerated, and the resulting collections are represented by **DirectInput8.GetDIDevices** and **DirectInputDevice8.GetDeviceObjectsEnum** objects.

Force-feedback effects are represented by the **DirectInputEffect** class. Methods of this interface are used to create, modify, start, and stop effects.

Interaction with Windows

Because Microsoft® DirectInput® works directly with the device drivers, it either suppresses or ignores Microsoft Windows® mouse and keyboard messages. It also ignores mouse and keyboard settings made by the user in Control Panel. It does, however, use the calibrations set for a joystick or other game controller.

DirectInput does not recognize keyboard character repeat settings. When using buffered data, DirectInput interprets each press and release as a single event with no repetition. When using immediate data, DirectInput is concerned only with the present physical state of the keys, not with keyboard events as interpreted by Windows.

DirectInput does not perform any character conversion or translation. For example, the SHIFT key is treated like any other key, not as a modifier of another keypress. Keys return the same identifiers regardless of the user's system language settings.

Under Windows 2000, acquiring the keyboard in exclusive mode prevents any applications that are subsequently launched from receiving keyboard data.

Because DirectInput works directly with the mouse driver, it bypasses the subsystem of Windows that interprets mouse data for windowed applications. Applications that rely on the Windows cursor for navigation should continue to use the standard Windows mouse messages and Microsoft® Win32® functions.

When using the system mouse in exclusive mode, DirectInput suppresses mouse messages, and therefore Windows is unable to show the standard cursor.

DirectInput ignores Control Panel settings such as acceleration and swapped buttons. However, DirectInput recognizes settings in the driver itself. For example, if the user has a three-button mouse and uses the driver-utility software to make the middle button a double-click shortcut, DirectInput reports a click of the middle button as two clicks of the primary button.

Using DirectInput

This section is a guide to implementing Microsoft® DirectInput®. For a broader overview, see Understanding DirectInput.

The following topics are discussed.

- Creating DirectInput
- DirectInput Device Enumeration
- DirectInput Devices
- DirectInput Device Data
- Action Mapping
- Force Feedback

Creating DirectInput

[C++]
The first step in any Microsoft® DirectInput® application is obtaining the **IDirectInput8** interface. You can do this most easily by calling the **DirectInput8Create** function.

You should create a single Microsoft® DirectInput object and not release it until the application terminates.

[Visual Basic]
The first step in any Microsoft® DirectInput® application is creating the **DirectInput8** object. Do this by using the **DirectX8.DirectInputCreate** method.

You should create a single **DirectInput8** object and not destroy it until the application terminates.

DirectInput Device Enumeration

Microsoft® DirectInput® can query the system for all available input devices, determine whether they are connected, and return information about them. This process is called enumeration.

[C++]
If your application is using only the standard keyboard or mouse, or both, you do not need to enumerate the available input devices. As explained under Creating a DirectInput Device, you can use predefined global variables when calling the **IDirectInput8::CreateDevice** method.

For all other input devices, and for systems with multiple keyboards or mouse devices, call **IDirectInput8::EnumDevices** or **IDirectInput8::EnumDevicesBySemantics** to enumerate available devices and find one that is suitable. You might also want to enumerate in order to give the user a choice of available devices.

The following is a sample implementation of the **IDirectInput8::EnumDevices** method:

```
/* lpdi is a valid IDirectInput8 interface pointer. */

GUID KeyboardGUID = GUID_SysKeyboard;

lpdi->EnumDevices(DIDEVTYPE_KEYBOARD,
                  DIEnumDevicesCallback,
                  &KeyboardGUID,
                  DIEDFL_ATTACHEDONLY);
```

The first parameter determines what types of devices are to be enumerated. It is NULL if you want to enumerate all devices, regardless of type; otherwise it is one of the DIDEVTYPE_* values described in **DIDEVICEINSTANCE**.

The second parameter is a pointer to a callback function to be called once for each device enumerated. This function can be declared by any name; the documentation uses the placeholder name **DIEnumDevicesCallback**.

The third parameter to the **EnumDevices** method is any 32-bit value that you want to pass into the callback function. In this example, it is a pointer to a variable of type **GUID**, passed in so that the callback can assign a keyboard instance identifier to this variable.

The fourth parameter is a flag to request enumeration of either all devices or only those that are attached (DIEDFL_ALLDEVICES or DIEDFL_ATTACHEDONLY).

If your application is using more than one input device, the callback function is a good place to initialize each device as it is enumerated. (For an example, see Tutorial 3: Using the Joystick.) You obtain the instance GUID of the device from the callback

function. You can also perform other processing here, such as looking for particular subtypes of devices or adding the device name to a list box.

The following sample callback function checks for the presence of an enhanced keyboard and stops the enumeration as soon as it finds one. It assigns the instance GUID of the last keyboard found to the *KeyboardGUID* variable (the address of which was passed in the *pvRef* variable by the call to **EnumDevices** in the previous example), which can then be passed to **IDirectInput8::CreateDevice**.

```
BOOL  hasEnhanced;

BOOL CALLBACK DIEnumKbdCallback(LPCDIDEVICEINSTANCE lpddi,
                        LPVOID pvRef)
{
  *(GUID*) pvRef = lpddi->guidInstance;
  if (GET_DIDEVICE_SUBTYPE(lpddi->dwDevType) ==
                  DIDEVTYPEKEYBOARD_PCENH)
  {
    hasEnhanced = TRUE;
    return DIENUM_STOP;
  }
  return DIENUM_CONTINUE;
} // End of callback
```

The first parameter points to a **DIDEVICEINSTANCE** structure containing information about the device. This structure is created by DirectInput.

The second parameter points to data passed in from **EnumDevices**. In this case, it is a pointer to the variable *KeyboardGUID*. This variable was assigned a default value earlier, but it is given a new value each time a device is enumerated. It is not important what instance GUID you use for a single keyboard, but the code does illustrate a technique for retrieving an instance GUID from the callback.

Rather than saving the instance GUID so that you can create the DirectInputDevice later, you can save the device created by DirectInput during enumeration. For more information, see Creating a DirectInput Device.

The return value in the example indicates that enumeration is to stop if the sought-for device has been found, or continue otherwise. Enumeration automatically stops as soon as all devices have been enumerated.

For more information on enumerating devices by using **IDirectInput8::EnumDevicesBySemantics**, see Action Mapping.

[Visual Basic]
If your application is using only the standard keyboard or mouse, or both, you do not need to enumerate the available input devices. See Creating a DirectInput Device.

For all other input devices, and for systems with multiple keyboards or mouse devices, call **DirectInput8.GetDIDevices** to build a collection of available devices. This method returns a **DirectInputEnumDevices8** object representing the collection. Each device in the collection can be retrieved as a **DirectInputDeviceInstance8** object by using the **DirectInputEnumDevices8.GetItem** method. The following example shows how to build a collection of game controllers and display them in a list box named *lstJoysticks*.

```
Dim diDevEnum As DirectInputEnumDevices8

Set diDevEnum = di.GetDIDevices(DI8DEVCLASS_GAMECTRL, DIEDFL_ATTACHEDONLY)

Dim i As Integer
For i = 1 To diDevEnum.GetCount
   Call lstJoysticks.AddItem(diDevEnum.GetItem(i).GetInstanceName)
Next i
```

At the very least, you must retrieve the unique identifier for a nonstandard device from the enumeration by calling **DirectInputDeviceInstance8.GetGuidInstance** before you can create a **DirectInputDevice8** object for that device. You might also want to enumerate devices in order to look for particular types and subtypes (by using **DirectInputDeviceInstance8.GetDevType**) or to populate a list box that enables the user to select a game controller.

You might even want to search for a device with particular capabilities. To do this, create a **DirectInputDevice8** object for each candidate in order to examine it further by using the **DirectInputDevice8.GetCapabilities** method.

DirectInput Devices

This section contains information about the code objects that represent devices such as a mouse, keyboards, or joysticks. The following topics are covered:

- Device Setup
- Creating a DirectInput Device
- Device Capabilities
- Cooperative Levels
- Device Object Enumeration
- Device Data Formats
- Device Properties
- Acquiring Devices

For information on how to retrieve and interpret data from devices, see DirectInput Device Data.

Device Setup

Your application must obtain a Microsoft® DirectInput® device object for each device from which it expects input. It must also prepare each device for use, which requires, at the very least, setting the data format and acquiring the device. You might also want to carry out other preparatory tasks such as getting information about the devices and changing their properties.

The following tasks are part of the setup process. Certain steps are always required. Other steps may be necessary if you need further information about devices or need to change default values.

1. Create the DirectInput device (required). See Creating a DirectInput Device.
2. Get the device capabilities (optional).
3. Enumerate the keys, buttons, and axes on the device (optional). See Device Object Enumeration.
4. Set the cooperative level (highly recommended).
5. Set the data format (required).
6. Set the device properties (you must at least set the buffer size if you intend to get buffered data).
7. When ready to read data, acquire the device (required). See Acquiring Devices.

Creating a DirectInput Device

To get input data from a device, you first have to create an object to represent that device.

[C++]
The **IDirectInput8::CreateDevice** method is used to obtain a pointer to the **IDirectInputDevice8** interface. Methods of this interface are then used to manipulate the device and obtain data.

The following code example, where *lpdi* is a pointer to the **IDirectInput8** interface, creates a keyboard device:

```
LPDIRECTINPUTDEVICE8  lpdiKeyboard;
lpdi->CreateDevice(GUID_SysKeyboard, &lpdiKeyboard, NULL);
```

The first parameter in **CreateDevice** is an instance GUID that identifies the instance of the device for which the interface is to be created. Microsoft® DirectInput® has two predefined GUIDs, *GUID_SysMouse* and *GUID_SysKeyboard*, which represent the system mouse and keyboard. You can pass one of these identifiers into the **CreateDevice** method. The global variable *GUID_Joystick* should not be used as a parameter for **CreateDevice** because it is a product GUID, not an instance GUID.

Note

If the computer has more than one mouse, input from all of them is combined to form the system device. The same is true for multiple keyboards.
DirectInput provides four other predefined GUIDs primarily for testing.

- GUID_SysKeyboardEm
- GUID_SysKeyboardEm2
- GUID_SysMouseEm
- GUID_SysMouseEm2.

Passing one of these GUIDs to **CreateDevice** grants access to the system keyboard or mouse through an emulation layer, at either level 1 or level 2. These GUIDs always represent the system mouse or keyboard. They are aliases for GUID_SysKeyboard and GUID_SysMouse, so they are not enumerated by **IDirectInput8::EnumDevices** or **IDirectInput8::EnumDevicesBySemantics** unless the DIEDFL_INCLUDEALIASES flag is passed.

For devices other than the system mouse or keyboard, use the instance GUID for the device returned by **EnumDevices** or **EnumDevicesBySemantics**. The instance GUID for a device is always the same. You can allow the user to select a device from a list of those enumerated, then save the GUID to a configuration file and use it again in future sessions.

[Visual Basic]
The **DirectInput8.CreateDevice** method is used to obtain a **DirectInputDevice8** object. Methods of this interface are then used to manipulate the device and obtain data.

The following code example, where *di* is the **DirectInput8** object, creates a keyboard device:

```
Dim diDev As DirectInputDevice
Set diDev = di.CreateDevice("GUID_SysKeyboard")
```

The parameter is an alias for a GUID that identifies the instance of the device for which the interface is to be created. Microsoft® DirectInput® provides two predefined GUIDs, represented by the strings *GUID_SysMouse* and *GUID_SysKeyboard*, which stand for the system mouse and keyboard. You can pass either of these to the **CreateDevice** method.

Note

If the computer has more than one mouse, input from all of them is combined to form the system device. The same is true for multiple keyboards.
For devices other than the system mouse or keyboard, use the instance GUID for the device obtained from **DirectInputDeviceInstance8.GetGuidInstance**. The instance GUID for a device is always the same. You can allow the user to select a device from a list of those enumerated, then save the GUID to a configuration file and use it again in future sessions.

In the following code example, it is presumed that the application has enumerated devices and found a suitable one, *diDevInstance*, which is to be created as a **DirectInputDevice**:

```
Dim guid As String
guid = diDevInstance.GetGuidInstance
Set diDev = di.CreateDevice(guid)
```

For more information on obtaining the **DirectInputDeviceInstance8** object, see DirectInput Device Enumeration.

Device Capabilities

Before you begin asking for input from a device, you need to know something about its capabilities. Does the joystick have a point-of-view hat? Is the mouse currently attached to the user's computer?

[C++]
Such questions are answered with a call to the **IDirectInputDevice8::GetCapabilities** method, which returns the data in a **DIDEVCAPS** structure. As with other such structures in Microsoft® DirectX®, you must initialize the **dwSize** member before passing this structure to the method.

Note

To optimize speed or memory usage, you can use the smaller **DIDEVCAPS_DX3** structure instead. For structure details, see Dinput.h in the Microsoft Software Development Kit (SDK).

The following code example checks whether the mouse is attached and whether it has a third axis (presumably a wheel). Assume that *lpdiMouse* is a valid Microsoft DirectInput® device.

```
DIDEVCAPS  DIMouseCaps;
HRESULT    hr;
BOOLEAN    WheelAvailable;

DIMouseCaps.dwSize = sizeof(DIDEVCAPS);
hr = lpdiMouse->GetCapabilities(&DIMouseCaps);
WheelAvailable = ((DIMouseCaps.dwFlags & DIDC_ATTACHED)
      && (DIMouseCaps.dwAxes > 2));
```

Another way to check for a button or axis is to call **IDirectInputDevice8::GetObjectInfo** for that object. If the call returns DIERR_OBJECTNOTFOUND, the object is not present. The following code determines whether there is a z-axis, even if it is not the third axis:

```
DIDEVICEOBJECTINSTANCE  didoi;

didoi.dwSize = sizeof(DIDEVICEOBJECTINSTANCE);
hr = lpdiMouse->GetObjectInfo(&didoi, DIMOFS_Z, DIPH_BYOFFSET);
WheelAvailable = SUCCEEDED(hr);
```

[Visual Basic]
Such questions are answered with a call to the **DirectInputDevice8.GetCapabilities** method, which returns the data in a **DIDEVCAPS** type.

The following code example checks whether the mouse is attached and whether it has a third axis (presumably a wheel):

```
/* diMouse is a valid DirectInputDevice object. */

Dim WheelAvailable As Boolean
Dim dicaps as DIDEVCAPS

Call diDev.GetCapabilities(dicaps)
WheelAvailable = ((dicaps.lFlags And DIDC_ATTACHED) _
      And (dicaps.lAxes > 2))
```

Another way to check for a certain button or axis is to call **DirectInputDevice8.GetObjectInfo** for that object. If the call raises the error DIERR_NOTFOUND, the object is not present. The following code determines whether there is a z-axis, even if it is not the third axis:

```
Dim didoi As DirectInputDeviceObjectInstance
On Error GoTo NOTFOUND
Set didoi = diDev.GetObjectInfo(DIMOFS_Z, DIPH_BYOFFSET)
On Error GoTo 0
.
.
.
NOTFOUND:
If Err.Number = DIERR_NOTFOUND Then MsgBox "No z-axis found."
End If
```

Cooperative Levels

[C++]
The cooperative level of a device determines how the input is shared with other applications and with the Microsoft® Windows® system. You set it by using the

IDirectInputDevice8::SetCooperativeLevel method, as in the following code example:

```
/* hwnd is the top-level window handle. */
lpdiDevice->SetCooperativeLevel(hwnd,
    DISCL_NONEXCLUSIVE | DISCL_FOREGROUND)
```

The parameters are the handle to the top-level window associated with the device (generally the application window) and one or more flags.

[Visual Basic]
The cooperative level of a device determines how the input is shared with other applications and with the Microsoft® Windows® system. You set it by using the **DirectInputDevice8.SetCooperativeLevel** method, as in the following code example:

```
'hwnd is the top-level window handle.
Call diDevice.SetCooperativeLevel(hWnd,
    DISCL_NONEXCLUSIVE Or DISCL_FOREGROUND)
```

The parameters are the handle to the top-level window associated with the device (generally the application window) and one or more flags.

Note

The **hWnd** property of a form does not become valid until the form is shown. If you are initializing the Microsoft® DirectInput® device in the **Load** method of the application's main form, you must call **Show** before attempting to set the cooperative level.

Although DirectInput provides a default setting, you should still explicitly set the cooperative level because this is the only way to give DirectInput the window handle. Without this handle, DirectInput cannot react to situations that involve window messages, such as joystick recalibration.

The valid flag combinations are shown in the following table.

Flags	Notes
DISCL_NONEXCLUSIVE DISCL_BACKGROUND	The default setting
DISCL_NONEXCLUSIVE DISCL_FOREGROUND	
DISCL_EXCLUSIVE DISCL_FOREGROUND	

DISCL_EXCLUSIVE DISCL_BACKGROUND	Not valid for keyboard or mouse

For the keyboard, you can also include DISCL_NOWINKEY in combination with DISCL_NONEXCLUSIVE. This flag disables the Windows logo key so that users cannot inadvertently break out of the application. In exclusive mode, the Windows logo key is always disabled.

Note

Even if the cooperative level for the application is disabling the Windows logo key passively through an exclusive cooperative level, or actively through use of the DISCL_NOWINKEY flag, that key will be active while the default action mapping UI is displayed.

The cooperative level has two main components: whether the device is being used in the foreground or the background, and whether it is being used exclusively or nonexclusively.

Foreground vs. Background

A foreground cooperative level means that the input device is available only when the application is in the foreground or, in other words, has the input focus. If the application moves to the background, the device is automatically unacquired, or made unavailable.

A background cooperative level really means foreground and background. A device with a background cooperative level can be acquired and used by an application at any time.

You will usually want to have foreground access only because most applications are not designed to respond to input that takes place when another program is in the foreground.

While developing an application, it is useful to employ conditional compilation so that the background cooperative level is always set for debugging. This prevents your application from losing access to the device every time it moves to the background as you switch to the debugging environment.

Exclusive vs. Nonexclusive

In most cases, the fact that your application is using a device at the exclusive level does not mean that other applications cannot get data from the device. However, it does mean that no other application can also acquire the device exclusively.

Consider the example of a music player that accepts input from a hand-held remote-control device, even when the application is running in the background. Now suppose you run a similar application that plays movies in response to signals from the same remote control. What happens when the user presses **Play**? Both programs start playing, which is probably not what the user wants. To prevent this from happening,

each application should have the DISCL_EXCLUSIVE flag set so that only one of them can be running at a time.

To use force-feedback effects, an application must have exclusive access to the device.

[C++]
Windows itself requires exclusive access to the system mouse. The reason is that system mouse events such as a click on an inactive window could force an application to unacquire the device, with potentially harmful results such as a loss of data from the input buffer. So when an application has exclusive access to the system mouse, created by passing GUID_SysMouse to **IDirectInput8::CreateDevice**, Windows is not allowed any access and no mouse messages are generated. A further side effect is that the cursor disappears.

[Visual Basic]
Windows itself requires exclusive access to the system mouse. The reason is that system mouse events such as a click on an inactive window could force an application to unacquire the device, with potentially harmful results such as a loss of data from the input buffer. So when an application has exclusive access to the system mouse, created by passing GUID_SysMouse to **DirectInput8.CreateDevice**, Windows is not allowed any access and no mouse messages are generated. A further side effect is that the cursor disappears.

When an application has exclusive access to the keyboard, DirectInput suppresses all keyboard messages except CTRL+ALT+DEL and, on Windows 95 and Windows 98, ALT+TAB. Under Windows 2000, no application launched subsequently to the exclusive-mode DirectInput application can get keyboard data.

Device Object Enumeration

It might be necessary for your application to determine what buttons or axes are available on a given device. To do this, you enumerate the device objects in much the same way as you enumerate devices.

Note

Because of limitations imposed by device drivers, enumeration of keyboard keys and indicator lights is not reliable. Objects might falsely be reported as present. You can get basic information about available keys from the device subtype, but there is no way to determine whether extra objects such as the menu key are available, other than asking the user for input.

[C++]
To some extent, **IDirectInputDevice8::EnumObjects** overlaps the functionality of **IDirectInputDevice8::GetCapabilities**. Either method can be used to determine how

many buttons or axes are available. However, **EnumObjects** is intended for cataloging all the available objects rather than checking for a particular one.

Like **IDirectInput8::EnumDevices**, the **EnumObjects** method has a callback function that enables you to do other processing on each object—for example, adding it to a list or creating a corresponding element on a user interface.

The following sample callback simply extracts the name of each object so that it can be added to a string list or array. This standard callback is documented under the placeholder name **DIEnumDeviceObjectsCallback**, but you can give it any name you like. Remember, this function is called once for each object enumerated.

```
char  szName[MAX_PATH];

BOOL CALLBACK DIEnumDeviceObjectsCallback(
                LPCDIDEVICEOBJECTINSTANCE lpddoi,
                LPVOID pvRef)
{
  lstrcpy(szName, lpddoi->tszName);
  // Now add szName to a list or array.
  .
  .
  .
  return DIENUM_CONTINUE;
}
```

The first parameter points to a structure containing information about the object. This structure is created for you by Microsoft® DirectInput®.

The second parameter is an application-defined pointer to data, equivalent to the second parameter to **EnumObjects**. In the example, this parameter is not used.

The return value in this case indicates that enumeration is to continue as long as there are objects to be enumerated.

Now here's the call to the **EnumObjects** method, which puts the callback function to work.

```
lpdiMouse->EnumObjects(DIEnumDeviceObjectsCallback,
                NULL, DIDFT_ALL);
```

The first parameter is the address of the callback function. The second parameter can be a pointer to any data you want to use or modify in the callback. The example does not use this parameter and so passes NULL. The third parameter is a flag to indicate which type or types of objects are to be included in the enumeration. In the example, all objects are to be enumerated. To restrict the enumeration, you can use one or more of the other DIDFT_* flags listed at **IDirectInputDevice8::EnumObjects**.

Note

Some of the DIDFT_* flags are combinations of others; for example, DIDFT_AXIS is equivalent to DIDFT_ABSAXIS | DIDFT_RELAXIS.

[Visual Basic]
You enumerate device objects by calling **DirectInputDevice8.GetDeviceObjectsEnum**, which returns an instance of the **DirectInputEnumDeviceObjects** class representing the collection of available device objects that match the requested parameters.

The following code example enumerates axes on a device:

```
' diDev is a DirectInputDevice object.

Dim diEnumObjects As DirectInputEnumDeviceObjects
Set diEnumObjects = diDev.GetDeviceObjectsEnum(DIDFT_AXIS)
```

The parameter is a combination of flags (just one, in this case) to indicate which type or types of objects to include in the enumeration.

Note
Some of the **CONST_DIDFTFLAGS** flags are combinations of others; for example, DIDFT_AXIS is equivalent to DIDFT_ABSAXIS **Or** DIDFT_RELAXIS.

To obtain information about a particular device object, call the methods of a **DirectInputDeviceObjectInstance** object obtained by calling **DirectInputEnumDeviceObjects.GetItem**. Information available for a device object includes its name, its type, and its offset in the data structure for the device.

The following code lists the names of the axes enumerated in the previous example:

```
Dim diDevObjInstance As DirectInputDeviceObjectInstance
Dim i As Integer
For i = 1 To diEnumObjects.GetCount
  Set diDevObjInstance = diEnumObjects.GetItem(i)
  Call List1.AddItem(diDevObjInstance.GetName)
Next i
```

Device Data Formats

If you are not using Action Mapping, setting the data format for a device is an essential step before you can acquire and begin using the device. This is true even if you do not intend to retrieve immediate (state) data from the device. Microsoft® DirectInput® uses the data format in many methods to identify particular device objects.

[C++]
The **IDirectInputDevice8::SetDataFormat** method tells DirectInput what device objects will be used and how the data will be arranged.

The examples in the reference for the **DIDATAFORMAT** and **DIOBJECTDATAFORMAT** structures show how to set up custom data formats for nonstandard devices. Fortunately, this step is not necessary for the joystick, keyboard, and mouse. DirectInput provides five global variables, *c_dfDIJoystick*, *c_dfDIJoystick2*, *c_dfDIKeyboard*, *c_dfDIMouse*, and *c_dfDIMouse2*, which can be passed in to **SetDataFormat** to create a standard data format for these devices.

In the following code example, *lpdiMouse* is an initialized pointer to a DirectInputDevice object:

```
lpdiMouse->SetDataFormat(&c_dfDIMouse);
```

Note

You cannot change the **dwFlags** member in the predefined **DIDATAFORMAT** global variables (for example, to change the property of an axis), because they are **const** variables. To change properties, use the **IDirectInputDevice8::SetProperty** method after setting the data format, but before acquiring the device.

[Visual Basic]
The **DirectInputDevice8.SetCommonDataFormat** and **DirectInputDevice8.SetDataFormat** methods tell Microsoft® DirectInput® what device objects will be used and how the data will be arranged.

For standard devices—the mouse, keyboard, and any game controller whose input data can be described in a **DIJOYSTATE** or **DIJOYSTATE2** type—you can set the data format by calling the **SetCommonDataFormat** method, passing in a constant from the **CONST_DICOMMONDATAFORMATS** enumeration. The common data formats are adequate for most applications.

For specialized devices, you must pass a description of the data format to the **SetDataFormat** method. The following code example sets the data format for a device with two axes, both of which require a **Long** for their data, and no buttons:

```
Dim dx As New DirectX8
Dim di As DirectInput8
Dim did As DirectInputDevice8
Dim fD As DIDATAFORMAT
Dim fDA(1) As DIOBJECTDATAFORMAT

Private Sub Form_Load()
   Set di = dx.DirectInputCreate()
```

```
    Set did = di.CreateDevice("GUID_SysMouse")

    fDA(0).lFlags = DIDOI_POLLED
    fDA(0).lOfs = 0
    fDA(0).lType = DIDFT_RELAXIS
    fDA(0).strGuid = "GUID_XAxis"

    fDA(1).lFlags = DIDOI_POLLED
    fDA(1).lOfs = 4
    fDA(1).lType = DIDFT_RELAXIS
    fDA(1).strGuid = "GUID_YAxis"

    fD.dataSize = 8
    fD.lFlags = DIDF_RELAXIS
    fD.lObjSize = 4
    fD.numObjs = 2

    did.SetDataFormat fD, fDA()

End Sub
```

Device Properties

[C++]

Properties of Microsoft® DirectInput® devices include the size of the data buffer, the range and granularity of values returned from an axis, whether axis data is relative or absolute, and the dead zone and saturation values for a joystick axis. Specialized devices can have other properties as well. For a list of properties defined by DirectInput, see **IDirectInputDevice8::GetProperty**.

With one exception—the gain property of a force-feedback device—properties can be changed only when the device is in an unacquired state.

Before calling the **IDirectInputDevice8::SetProperty** or the **IDirectInputDevice8::GetProperty** method you must set up a property structure that consists of a **DIPROPHEADER** structure and one or more elements for data. There is potentially a great variety of properties for input devices, and **SetProperty** must be able to work with all sorts of structures defining those properties. The purpose of the **DIPROPHEADER** structure is to define the size of the property structure and how the data is to be interpreted.

DirectInput includes the following predefined property structures:

- **DIPROPDWORD** defines a structure containing a **DIPROPHEADER** and a **DWORD** data member for properties that require a single value, such as a buffer size.
- **DIPROPRANGE** is for range properties, which require two values (maximum and minimum). It consists of a **DIPROPHEADER** and two **LONG** data members.
- **DIPROPGUIDANDPATH** is a specialized property structure enabling applications to perform operations on a Human Interface Device (HID) that are not supported by DirectInput. The structure consists of a **DIPROPHEADER**, a **GUID**, and a Unicode string for the path.
- **DIPROPSTRING** is for Unicode string properties. The structure comprises a **DIPROPHEADER** and a Unicode string.

For **SetProperty**, the data members of the property structure are the values that you want to set. For **GetProperty**, the current value is returned in these members.

Before the call to **GetProperty** or **SetProperty**, the **DIPROPHEADER** structure must be initialized with the following:

- The size of the property structure.
- The size of the **DIPROPHEADER** structure itself.
- An object identifier.
- A value indicating how the object identifier should be interpreted.

When getting or setting properties for a whole device, the object identifier **dwObj** is 0, and the **dwHow** member is DIPH_DEVICE. If you want to get or set properties for a device object (for example, a particular axis), the combination of **dwObj** and **dwHow** values identifies the object. For details, see **DIPROPHEADER**.

After setting up the property structure, pass the address of its header into **GetProperty** or **SetProperty**, along with an identifier for the property that you want to obtain or change.

The following values are used to identify the property passed to **SetProperty** and **GetProperty.** For more information, see **IDirectInputDevice8::GetProperty**.

- DIPROP_BUFFERSIZE. See also Buffered and Immediate Data. The buffer size can also be set by using the **IDirectInputDevice8::SetActionMap** method.
- DIPROP_AXISMODE. See also Relative and Absolute Axis Coordinates.
- DIPROP_CALIBRATIONMODE
- DIPROP_GRANULARITY
- DIPROP_FFGAIN
- DIPROP_FFLOAD
- DIPROP_AUTOCENTER
- DIPROP_RANGE

- DIPROP_DEADZONE
- DIPROP_SATURATION

For more information about the last three properties, see also Interpreting Joystick Axis Data.

The following code example sets the buffer size for a device to hold up to 10 data items:

```
DIPROPDWORD  dipdw;
HRESULT      hr;
dipdw.diph.dwSize = sizeof(DIPROPDWORD);
dipdw.diph.dwHeaderSize = sizeof(DIPROPHEADER);
dipdw.diph.dwObj = 0;
dipdw.diph.dwHow = DIPH_DEVICE;
dipdw.dwData = 10;
hr = lpdiDevice->SetProperty(DIPROP_BUFFERSIZE, &dipdw.diph);
```

[Visual Basic]
Properties of Microsoft® DirectInput® devices include the size of the data buffer, the range and granularity of values returned from an axis, whether axis data is relative or absolute, and the dead zone and saturation values for a joystick axis. Specialized devices can have other properties as well. For a list of properties defined by DirectInput, see **DirectInputDevice8.GetProperty**.

With one exception—the gain property of a force-feedback device—properties can be changed only when the device is in an unacquired state.

The **DirectInputDevice8.SetProperty** and **DirectInputDevice8.GetProperty** methods take two parameters: a GUID alias in string form that identifies the property being set, and data of type **Any**. The data is actually passed in one of the following types:

- **DIPROPLONG** is for properties that require a single value, such as a buffer size. The property data consists of a single **Long**.
- **DIPROPRANGE** is for range properties, which require two values (maximum and minimum). The property data consists of two **Long** data members.

For **SetProperty**, the data members of the property types are the values you want to set. For **GetProperty**, the current value is returned in these members.

In addition to the actual property data, both these types contain two other members: **lHow** and **lObj**.

The values in **lHow** and **lObj** work together, with **lHow** signifying the system that is used to identify the device object whose property is being set or retrieved, and **lObj** identifying the device object.

If **lHow** is DIPH_BYID, the device object is described by a unique numerical identifier in **lObj**. This ID can be extracted from the value returned by **DirectInputDeviceObjectInstance.GetType** after device objects have been enumerated.

For most applications, it is simpler to identify the device object by its offset within the data structure established by **DirectInputDevice8.SetCommonDataFormat** or **DirectInputDevice8.SetDataFormat**. In this case, **lHow** is DIPH_BYOFFSET, and **lObj** is the offset, in bytes. For the keyboard, mouse, and any game controller whose data can be returned in a **DIJOYSTATE** or **DIJOYSTATE2** type, the device object can be identified by a predefined constant. See **CONST_DIKEYFLAGS**, **CONST_DIMOUSEOFS**, and **CONST_DIJOYSTICKOFS**.

The **lHow** member can also contain DIPH_DEVICE, which means that the property belongs to the entire device, rather than a single device object. Buffer size is an example of such a property. When **lHow** is DIPH_DEVICE, **lObj** is 0.

The following strings are used to identify the property passed to **SetProperty** and **GetProperty.** For more information, see **DirectInputDevice8.GetProperty**.

- DIPROP_AXISMODE. See also Relative and Absolute Axis Coordinates.
- DIPROP_BUFFERSIZE. See also Buffered and Immediate Data.
- DIPROP_DEADZONE
- DIPROP_GRANULARITY
- DIPROP_RANGE
- DIPROP_SATURATION

For more information about dead zone, range, and saturation, see also Interpreting Joystick Axis Data.

The following code example sets the buffer size for a device:

```
' diDev is a DirectInputDevice whose data format has been set.

Dim diProp As DIPROPLONG
diProp.lHow = DIPH_DEVICE
diProp.lObj = 0
diProp.lData = 10
Call diDev.SetProperty("DIPROP_BUFFERSIZE", diProp)
```

Acquiring Devices

Acquiring a Microsoft® DirectInput® device means giving your application access to it. As long as a device is acquired, DirectInput is making its data available to your application. If the device is not acquired, you can manipulate its characteristics but not obtain any data.

Acquisition is not permanent. Your application might acquire and unacquire a device many times.

In certain cases, depending on the cooperative level, a device is unacquired automatically whenever the application moves to the background. The mouse is automatically unacquired when the user clicks a menu, because at this point Microsoft Windows® assumes control of the device.

You must unacquire a device before changing its properties. The only exception is that you can change the gain for a force-feedback device while it is in an acquired state.

The acquisition mechanism is needed for two reasons:

First, DirectInput must be able to tell the application when the flow of data from the device has been interrupted by the system. For instance, if the user has switched to another application with ALT+TAB and has used the input device in that application, your application needs to know that the input no longer belongs to it and that the state of the buffers might have changed. Or consider an application with the DISCL_FOREGROUND cooperative level. The user holds down the SHIFT key and switches to another application. Then the user releases the key and switches back to the first application. As far as the first application is concerned, the SHIFT key is still down. The acquisition mechanism, by telling the application that input was lost, enables it to recover from these conditions.

Second, because your application can alter the properties of the device, without safeguards DirectInput would have to check the properties each time you wanted to retrieve data. This would be very inefficient. Even worse, a potential disaster could happen such as a hardware interrupt accessing a data buffer just as you were changing the buffer size. Therefore DirectInput requires your application to unacquire the device before changing properties. When you reacquire it, DirectInput checks the properties and decides on the optimal way of transferring data from the device to the application. This is done only once, so the data retrieval methods can be very fast.

[C++]
Because the most common cause of losing a device is that your application moves to the background, you may want to reacquire devices whenever your application is activated. Be careful, though, about relying on a WM_ACTIVATE handler at startup time. The first WM_ACTIVATE message will likely arrive when your window is being initialized before DirectInput has been set up. To ensure that the device is acquired at startup, call **IDirectInputDevice8::Acquire** as soon as the device has been initialized.

Even acquiring the device on activation of your program window might not cover all cases in which a device is unacquired, especially for devices other than the standard mouse or keyboard. Because your application might unacquire a device unexpectedly, you need a mechanism for checking the acquisition state before attempting to get data from the device. The Scrawl Sample does this in the **OnMouseInput** function, in which a DIERR_INPUTLOST error triggers a message to reacquire the mouse. (See also Tutorial 2: Using the Mouse.)

[Visual Basic]
Because your application might unacquire a device unexpectedly, especially if you have set the exclusive cooperative level, you must ensure that the application tracks the state of acquisition. One technique is to check for the DIERR_INPUTLOST error after attempting to retrieve data. If this error is raised, you know the device has been unacquired. If your application is getting input in response to event notification, an event is signaled when acquisition is lost.

See the ScrawlB sample for more information about how to manage device acquisition in exclusive mode.

Attempting to reacquire a device that is already acquired does no harm. Redundant calls to **Acquire** are ignored, and the device can always be unacquired with a single call to **Unacquire**.

[C++]
Windows does not have access to the mouse when your application is using it in exclusive mode. If you want Windows to acquire the mouse, you must release it. The Scrawl Sample responds to a right-click by unacquiring the mouse, putting the Windows cursor in the same spot as its own, displaying a shortcut menu, and letting Windows handle the input until a menu choice is made.

[Visual Basic]
Windows does not have access to the mouse when your application is using it in exclusive mode. If you want Windows to acquire the mouse, you must release it. The ScrawlB sample responds to a right-click button by unacquiring the mouse, putting the Windows cursor in the same spot as its own, displaying a shortcut menu, and letting Windows handle the input until a menu choice is made.

DirectInput Device Data

This section covers the following basic concepts of getting data from Microsoft® DirectInput® devices.

- Buffered and Immediate Data
- Time Stamps and Sequence Numbers
- Polling and Event Notification

- Relative and Absolute Axis Coordinates

The following topics describe specific details about output data and mouse, keyboard, and joystick input data.

- Mouse Data
- Keyboard Data
- Joystick Data
- Output Data

DirectInput provides two mechanisms for identifying data retrieved from devices. Using the traditional mechanism, the application identifies the data as coming from a particular device object, such as a the x-axis on a mouse or the F1 key on a keyboard. With the newer system, available for the first time in Microsoft DirectX® 8.0, DirectInput associates application-defined values (for instance, identifiers for particular game actions such as moving or shooting) with the most suitable device objects available. When data is retrieved, the application does not concern itself with the actual device or device object from which the data comes. It acts only on the meaning of that data.

Most of the information in this section applies to both mechanisms. For more information on the newer way of identifying device data, see Action Mapping.

Buffered and Immediate Data

Microsoft® DirectInput® supplies two types of data: buffered and immediate. Buffered data is a record of events that are stored until an application retrieves them. Immediate data is a snapshot of the current state of a device.

You might use immediate data in an application that is concerned only with the current state of a device—for example, a flight combat simulation that responds to the current position of the joystick and the state of one or more buttons. Buffered data might be the better choice where events are more important than states—for example, in an application that responds to movement of the mouse and button clicks. You can also use both types of data, as you might, for example, if you wanted to get immediate data for joystick axes but buffered data for the buttons.

Note

If you are using Action Mapping, you will want to retrieve buffered data because the buffered data packets contain the application-defined data associated with the input. Doing so means that you are responsible for tracking the position of absolute axes, since events but not states are reported in buffered data packets.

[C++]
An application retrieves immediate data by calling the **IDirectInputDevice8::GetDeviceState** method. As the name implies, this method returns the current state of the device—for example, whether each button is up or down. The method provides no data about what has happened with the device since

the last call, apart from implicit information that you can derive by comparing the current state with the last one. For example, if a user presses and releases a button between two calls to **GetDeviceState**, your application will never be given this input. On the other hand, if the user is holding a button down, **GetDeviceState** continues reporting that fact until the user releases it.

This way of reporting the device state is different from the way Microsoft Windows® reports events with one-time messages such as WM_LBUTTONDOWN. It is more like the results from the Microsoft Win32® **GetKeyboardState** function. If you are polling a device with **GetDeviceState**, you are responsible for determining what constitutes a button click, a double-click, a single keystroke, and so on, and for ensuring that your application does not keep responding to a button-down or key-down state when it's not appropriate to do so.

With buffered data, events are stored until you are ready to deal with them. Every time a button or key is pressed or an axis is moved, information about the event is placed in a **DIDEVICEOBJECTDATA** structure in the buffer. If the buffer overflows, new data is lost. Your application reads the buffer with a call to **IDirectInputDevice8::GetDeviceData**. You can read any number of items at a time.

Reading an item normally deletes it from the buffer, but you also have the choice of peeking without deleting.

To get buffered data, you must first set the buffer size with the **IDirectInputDevice8::SetProperty** or **IDirectInputDevice8::SetActionMap** method. (See the example under Device Properties.) Set the buffer size before acquiring the device for the first time. For reasons of efficiency, the default size of the buffer is 0, and you cannot obtain buffered data unless you change this value. The size of the buffer is measured in items of data for that type of device, not in bytes.

Check the value of the *pdwInOut* parameter after a call to the **GetDeviceData** method. The number of items retrieved from the buffer is returned in this variable.

Note

For devices that do not generate interrupts, such as analog joysticks, DirectInput does not obtain any data until you call the **IDirectInputDevice8::Poll** method. For more information, see Polling and Event Notification.

For examples of retrieving buffered data, see **IDirectInputDevice8::GetDeviceData**.

[Visual Basic]
An application retrieves immediate data by calling one of the following methods:

- **DirectInputDevice8.GetDeviceStateKeyboard**. For devices that retrieve data in a **DIKEYBOARDSTATE** type. (For this and the following three methods, the data format must have been set by using **DirectInputDevice8.SetCommonDataFormat**.)
- **DirectInputDevice8.GetDeviceStateMouse**. For devices that retrieve data in a **DIMOUSESTATE** type.

- **DirectInputDevice8.GetDeviceStateMouse2**. For devices that retrieve data in a **DIMOUSESTATE2** type.
- **DirectInputDevice8.GetDeviceStateJoystick**. For devices that retrieve data in a **DIJOYSTATE** type.
- **DirectInputDevice8.GetDeviceStateJoystick2**. For devices that retrieve data in a **DIJOYSTATE2** type.
- **DirectInputDevice8.GetDeviceState**. For devices that use custom data formats, as set by using **DirectInputDevice8.SetDataFormat**.

As the names imply, each of these methods returns the current state of the device—for example, whether each button is up or down. The method provides no data about what has happened with the device since the last call, apart from implicit information that you can derive by comparing the current state with the last one. For example if a user presses and releases a button between two calls to the method, your application will never be given this input. On the other hand, if the user is holding a button down, the method continues reporting that fact until the user releases it.

This way of reporting the device state is different from the way Microsoft Visual Basic® reports events with one-time events such as **Click** and **Keydown**. If you are polling a device with one of the **GetDeviceState** methods, you are responsible for determining what constitutes a button click, a double-click, a single keystroke, and so on, and for ensuring that your application does not keep responding to a button-down or key-down state when it is not appropriate to do so.

With buffered data, events are stored until you are ready to deal with them. Every time a button or key is pressed or an axis is moved, information about the event is placed in a **DIDEVICEOBJECTDATA** type in the buffer. Your application reads the buffer with a call to **DirectInputDevice8.GetDeviceData**. You can read any number of items at a time.

Reading an item normally deletes it from the buffer, but you also have the choice of retrieving without deleting by setting the DIGDD_PEEK flag.

To get buffered data, you must first set the buffer size by using the **DirectInputDevice8.SetProperty** or DirectInputDevice8.SetActionMap method. (See the example under Device Properties.) Set the buffer size before acquiring the device for the first time. For reasons of efficiency, the default size of the buffer is 0, and you cannot obtain buffered data unless you change this value. The size of the buffer is measured in items of data for that type of device, not in bytes.

The return value of **GetDeviceData** tells you the number of items retrieved from the buffer. If the buffer has overflowed, no data is returned, and **GetDeviceData** raises an error, which the application should trap.

Note

For devices that do not generate interrupts, such as analog joysticks, DirectInput does not obtain any data until you call the **DirectInputDevice8.Poll** method. For more information, see Polling and Event Notification.

See also:

- Time Stamps and Sequence Numbers
- Mouse Data
- Keyboard Data
- Joystick Data

Time Stamps and Sequence Numbers

[C++]
When Microsoft® DirectInput® input data is buffered (see Buffered and Immediate Data), each **DIDEVICEOBJECTDATA** structure contains not only information about the type of event and the device object associated with it. It also contains a time stamp and a sequence number.

The **dwTimeStamp** member contains the system time, in milliseconds, at which the event took place. This is equivalent to the value that would have been returned by the Microsoft Win32® **GetTickCount** function, but at a higher resolution.

The **dwSequence** member contains a sequence number assigned by DirectInput. The DirectInput system keeps a single sequence counter, which is incremented by each buffered event from any device. (Simultaneous events on the same device, such as diagonal mouse movements, increment the count only once.) Use this number to compare events from different devices and see which came first. The **DISEQUENCE_COMPARE** macro, used for comparing DirectInput sequence numbers, takes wraparound into account.

[Visual Basic]
When Microsoft® DirectInput® input data is buffered (see Buffered and Immediate Data), each **DIDEVICEOBJECTDATA** type contains not only information about the type of event and the device object associated with it. It also contains a time stamp and a sequence number.

The **lTimeStamp** member contains the system time, in milliseconds, at which the event took place. This is equivalent to the value that would have been returned by the Microsoft Win32® **GetTickCount** function, but at a higher resolution.

The **lSequence** member contains a sequence number assigned by DirectInput. The DirectInput system keeps a single sequence counter, which is incremented by each buffered event from any device. (Simultaneous events on the same device, such as diagonal mouse movements, increment the count only once.) Use this number to compare events from different devices and see which came first.

Note
Events are always placed in the buffer in chronological order, so you don't need to check the sequence numbers to sort the events from a single device.

Polling and Event Notification

There are two ways to find out whether input data is available: polling and event notification.

Polling a device means regularly getting the current state of the device objects or checking the contents of the event buffer. Polling is typically used by real-time games that are never idle, but instead are constantly updating and rendering the game world.

Polling

[C++]
In a C++ application, polling would typically be done within the message loop.

Some joysticks and other game devices, or particular objects on them, do not generate hardware interrupts and do not return any data or signal any events until you call the **IDirectInputDevice8::Poll** method. (This behind-the-scenes polling is not to be confused with the kind of application polling just discussed. **Poll** does not retrieve any data, but merely makes data available.)

To find out whether it is necessary to call **Poll** each time you want to retrieve data, first set the data format for the device, then call the **IDirectInputDevice8::GetCapabilities** method and check for the DIDC_POLLEDDATAFORMAT flag in the **DIDEVCAPS** structure.

Do not confuse the DIDC_POLLEDDATAFORMAT flag with the DIDC_POLLEDDEVICE flag. The latter is set if any object on the device requires polling, regardless of data format. You can then find out whether this is the case for a particular object by calling the **IDirectInputDevice8::GetObjectInfo** method and checking for the DIDOI_POLLED flag in the **DIDEVICEOBJECTINSTANCE** structure.

The DIDC_POLLEDDEVICE flag describes the worst case for the device, not the actual situation. For example, a Human Interface Device (HID) mouse with software-controllable resolution might be marked as DIDC_POLLEDDEVICE because reading the resolution information requires polling. If you want to retrieve only the standard button and axis data, polling the device under these conditions is not necessary. However, there is no harm in calling the **IDirectInputDevice8::Poll** method for any input device. If the call is unnecessary, it has no effect and is very fast.

[Visual Basic]
In a Microsoft® Visual Basic® application, polling would typically be done in the **Sub Main** procedure.

Some joysticks and other game devices, or particular objects on them, do not generate hardware interrupts and do not return any data or signal any events until you call the **DirectInputDevice8.Poll** method. (This behind-the-scenes polling is not to be confused with the kind of application polling just discussed. **Poll** does not retrieve any data, but merely makes data available.)

To find out whether it is necessary to call **Poll** each time you want to retrieve data, first set the data format for the device, then call the **DirectInputDevice8.GetCapabilities** method and check for the DIDC_POLLEDDATAFORMAT flag in the **DIDEVCAPS** type.

Do not confuse the DIDC_POLLEDDATAFORMAT flag with the DIDC_POLLEDDEVICE flag. The latter is set if any object on the device requires polling. You can then find out whether this is the case for a particular object by calling the **DirectInputDevice8.GetObjectInfo** method to get a **DirectInputDeviceObjectInstance** object, and then checking for the DIDOI_POLLED flag in the valued returned by **DirectInputDeviceObjectInstance.GetFlags**.

The DIDC_POLLEDDEVICE flag describes the worst case for the device, not the actual situation. For example, an HID mouse with software-controllable resolution might be marked as DIDC_POLLEDDEVICE because reading the resolution information requires polling. If you want to retrieve only the standard button and axis data, polling the device under these conditions is not necessary. However, there is no harm in calling the **Poll** method for any input device. If the call is unnecessary, it has no effect and is very fast.

Event Notification

[C++]
Event notification is suitable for applications such as the Scrawl Sample that wait for input before doing anything.

To use event notification, set up a thread-synchronization object with the Microsoft® Win32® **CreateEvent** function, and then associate this event with the device by passing its handle to the **IDirectInputDevice8::SetEventNotification** method. The event is then signaled by Microsoft DirectInput® whenever the state of the device changes. Your application can receive notification of the event with a Win32 function such as **WaitForSingleObject**, and then respond by checking the input buffer to find out what the event was. For code examples, see the Scrawl sample and **IDirectInputDevice8::SetEventNotification**.

[Visual Basic]
Event notification is suitable for applications such as the ScrawlB Sample that wait for input before doing anything.

To use event notification, implement **DirectXEvent8** in the form or module in which you want to retrieve data. Then create an event handle by using **DirectX8.CreateEvent**, and pass this handle to **DirectInputDevice8.SetEventNotification**. Now, whenever an input event occurs on the device, the **DirectXEvent8.DXCallback** method is called, and in your implementation of this method you can retrieve either the device state or buffered data just as you would if you were doing so in **Sub Main**.

Relative and Absolute Axis Coordinates

Axis coordinates can be returned as relative values—that is, the amount by which they have changed since the application last retrieved the device state or, in the case of buffered input, since the last item was put in the buffer.

Absolute axis coordinates are a running total of all the relative coordinates returned by the system since the device was acquired. In other words, they show the position of the axis in relation to a fixed point.

By default, mouse axes are reported as relative coordinates and joystick axes as absolute coordinates. You can change the coordinate system for a device by setting a property. For more information, see Device Properties.

Mouse Data

[C++]
To set up the mouse device for data retrieval, first call the **IDirectInputDevice8::SetDataFormat** method with the *c_dfDIMouse* or *c_dfDIMouse2* global variable as the parameter value. Use *c_dfDIMouse2* if you want to support more than four mouse buttons.

For maximum performance in a full-screen application, set the cooperative level to DISCL_EXCLUSIVE | DISCL_FOREGROUND. Note that the exclusive setting causes the Microsoft® Windows® cursor to disappear. The DISCL_FOREGROUND setting causes the application to lose access to the mouse when you switch to a debugging window. Changing to DISCL_BACKGROUND enables you to debug the application more easily, but at a cost in performance.

[Visual Basic]
To set up the mouse device for data retrieval, first call the **DirectInputDevice8.SetCommonDataFormat** method with DIFORMAT_MOUSE as the parameter value.

For maximum performance in a full-screen application, set the cooperative level to DISCL_EXCLUSIVE **Or** DISCL_FOREGROUND. Note that the exclusive setting causes the Microsoft® Windows® cursor to disappear. The DISCL_FOREGROUND

setting causes the application to lose access to the mouse when you switch to the Microsoft Visual Basic® development environment. Changing to DISCL_BACKGROUND enables you to debug the application more easily, but at a cost in performance.

The following sections give more information about retrieval and interpretation of immediate and buffered mouse data:

- Immediate Mouse Data
- Buffered Mouse Data
- Interpreting Mouse Axis Data
- Checking for Lost Mouse Input

See Also

Device Data Formats, Cooperative Levels

Immediate Mouse Data

[C++]
To retrieve the current state of the mouse, call **IDirectInputDevice8::GetDeviceState** with a pointer to a **DIMOUSESTATE** or a **DIMOUSESTATE2** structure, depending on the data format. The mouse state returned in the structure includes axis data and the state of each of the buttons.

The first three members of the structure hold the axis coordinates. (See Interpreting Mouse Axis Data.)

The **rgbButtons** member is an array of bytes, one for each of four or eight buttons. For a traditional mouse, the first element in the array is generally the left button, the second is the right button, and the third is the middle button. The high bit is set if the button is down, and it is clear if the button is up or not present.

[Visual Basic]
To retrieve the current state of the mouse, call **DirectInputDevice8.GetDeviceStateMouse** or **DirectInputDevice8.GetDeviceStateMouse2**, passing in a **DIMOUSESTATE** or **DIMOUSESTATE2** type. The mouse state returned in the type includes axis data and the state of each of the buttons.

The **x**, **y,** and **z** members of the **DIMOUSESTATE** and **DIMOUSESTATE2** types hold the axis coordinates. (See Interpreting Mouse Axis Data.) The **buttons** member is an array of bytes, one for each button. The first element in the array is generally the left button, the second is the right button, the third is the middle button. The high bit is set if the button is down, and it is clear if the button is up or not present.

See also

Buffered and Immediate Data

Buffered Mouse Data

To retrieve buffered data from the mouse, you must first set the buffer size (see Device Properties). The default size of the buffer is 0, so this step is essential.

[C++]
You must also declare an array of **DIDEVICEOBJECTDATA** structures. This array can have up to the same number of elements as the buffer size. You do not have to retrieve the entire contents of the buffer with a single call. You can have just one element in the array and retrieve events one at a time until the buffer is empty.

After acquiring the device, you can examine and flush events in the buffer at any time by using the **IDirectInputDevice8::GetDeviceData** method. (See Buffered and Immediate Data.) On return, each element in the **DIDEVICEOBJECTDATA** array represents a change in state for a single object on the mouse. For example, if the user presses button 0 and moves the mouse diagonally, the array passed to **GetDeviceData** (if it has at least three elements, and *pdwInOut* is at least 3) will have three elements filled in—an element for button 0 being pressed, an element for the change in the x-axis, and an element for the change in the y-axis—and the value of *pdwInOut* will be set to 3.

You can determine which object an element in the array refers to by checking the **dwOfs** member of the **DIDEVICEOBJECTDATA** structure against the values returned by the following macros:

- **DIMOFS_BUTTON0** *to* **DIMOFS_BUTTON7**
- **DIMOFS_X**
- **DIMOFS_Y**
- **DIMOFS_Z**

Each of these values is derived from the offset of the data for the object in a **DIMOUSESTATE** or **DIMOUSESTATE2** structure. For example, **DIMOFS_BUTTON0** is equivalent to the offset of **rgbButtons[0]** in the **DIMOUSESTATE** structure. **DIMOFS_BUTTON4** to **DIMOFS_BUTTON7** are supported only for **DIMOUSESTATE2**. With the macros you can use simple comparisons to determine which device object is associated with an item in the buffer. For example:

```
DIDEVICEOBJECTDATA  *lpdidod;
int             n;
.
.
```

```
.
/* MouseBuffer is an array of DIDEVICEOBJECTDATA structures
   that has been set by a call to GetDeviceData.
   n is incremented in a loop that examines all filled elements
   in the array. */
lpdidod = &MouseBuffer[n];
if (( (int) lpdidod->dwOfs == DIMOFS_BUTTON0)
     && (lpdidod->dwData & 0x80))
{
 ;  // Do something in response to left button press.
}
```

The data for the change of state of the device object is located in the **dwData** member of the **DIDEVICEOBJECTDATA** structure. For axes, the coordinate value is returned in this member. For button objects, only the low byte of **dwData** is significant. The high bit of this byte is set if the button was pressed, and it is clear if the button was released. In other words, the button was pressed if (**dwData** & 0x80) is nonzero.

If you are using Action Mapping, ignore the value in **dwOfs** and instead retrieve the application-defined data associated with the event from the **uAppData** member.

For more information on the other members of the **DIDEVICEOBJECTDATA** structure, see Time Stamps and Sequence Numbers.

[Visual Basic]
You must also declare an array of **DIDEVICEOBJECTDATA** types. This array can have up to the same number of elements as the buffer size. You do not have to retrieve the entire contents of the buffer with a single call. You can have just one element in the array and retrieve events one at a time until the buffer is empty.

After acquiring the device, you can examine and flush the buffer at any time by using the **DirectInputDevice8.GetDeviceData** method. (See Buffered and Immediate Data.) On return, each element in the **DIDEVICEOBJECTDATA** array represents a change in state for a single object on the mouse. For instance, if the user presses button 0 and moves the mouse diagonally, the array passed to **GetDeviceData** (if it has at least three elements) will have three elements filled in—an element for button 0 being pressed, an element for the change in the x-axis, and an element for the change in the y-axis—and the return value of the method will be 3.

You can determine which object an element in the array refers to by checking the **lOfs** member of the **DIDEVICEOBJECTDATA** type against the constants in the **CONST_DIMOUSEOFS** enumeration. Each of these values is derived from the offset of the data for the object in a **DIMOUSESTATE** or **DIMOUSESTATE2** type. For example, DIMOFS_BUTTON0 is equivalent to the offset of **buttons(0)** in the **DIMOUSESTATE** type. **DIMOFS_BUTTON4** to **DIMOFS_BUTTON7** are supported only for **DIMOUSESTATE2**.

The data for the change of state of the device object is located in the **lData** member of the **DIDEVICEOBJECTDATA** type. For axes, the coordinate value is returned in this member. For button objects, only the low byte of **lData** is significant. The high bit of this byte is set if the button was pressed, and it is clear if the button was released. In other words, the button was pressed if (**lData And** &H80) is nonzero.

For more information on the other members of the **DIDEVICEOBJECTDATA** type, see Time Stamps and Sequence Numbers.

The following code example retrieves the entire contents of the buffer (which contains *BufferSize* elements) and responds to various events:

```
' objDIDev is a DirectInputDevice object.
Dim diDeviceData(1 To BufferSize) As DIDEVICEOBJECTDATA
Dim NumEvents As Integer
Dim i As Integer

NumEvents = objDIDev.GetDeviceData(diDeviceData, 0)
For i = 1 To NumEvents
   Select Case diDeviceData(i).lOfs
      Case DIMOFS_X
        ' Respond to x-axis movement.

      Case DIMOFS_Y
        ' Respond to y-axis movement.

      Case DIMOFS_BUTTON0
        If diDeviceData(i).lData And &H80 Then
          ' Respond to left button pressed.
        Else
          ' Respond to left button released.
        End If

   End Select
Next i
```

Interpreting Mouse Axis Data

The data returned for the x-axis and y-axis of a mouse indicates the movement of the mouse itself, not the cursor. The units of measurement are based on the values returned by the mouse hardware and have nothing to do with pixels or any other form of screen measurement. Because Microsoft® DirectInput® communicates directly with the mouse driver, the values for mouse speed and acceleration set by the user in Control Panel do not affect this data.

Axis data returned from the mouse can be either relative or absolute. (See Relative and Absolute Axis Coordinates.) Because a mouse is a relative device—unlike a joystick, it does not have a home position—relative data is returned by default.

[C++]
The axis mode, which specifies whether relative or absolute data should be returned, is a property that can be changed before the device is acquired. (See Device Properties.) To set the axis mode to absolute, call **IDirectInputDevice8::SetProperty** with the DIPROP_AXISMODE value in the *rguidProp* parameter and with DIPROPAXISMODE_ABS in the **dwData** member of the **DIPROPDWORD** structure.

[Visual Basic]
The axis mode, which specifies whether relative or absolute data should be returned, is a property that can be changed before the device is acquired. (See Device Properties.) To set the axis mode to absolute, call **DirectInputDevice8.SetProperty** with "DIPROP_AXISMODE" in the *guid* parameter and with DIPROPAXISMODE_ABS in the **lData** member of the **DIPROPLONG** type.

When the axis mode for the mouse is set to relative, the axis coordinate represents the number of units that the device has been moved along the axis since the last value was returned. A negative value indicates that the mouse was moved to the left for the x-axis, or away from the user for the y-axis, or that the z-axis (the wheel) was rotated toward the user. Positive values indicate movement in the opposite direction.

When the axis mode is set to absolute, the coordinates are simply a running total of all relative motions received by DirectInput. The axis coordinates are not initialized to any particular value when the device is acquired, so your application should treat absolute values as relative to an unknown origin. You can record the current absolute position whenever the device is acquired and save it as the virtual origin. This virtual origin can then be subtracted from subsequent absolute coordinates retrieved from the device to compute the relative distance that the mouse has moved from the point of acquisition.

The data returned for the axis coordinates is also affected by the granularity property of the device. For the x-axis and y-axis of the mouse, granularity is normally 1. In other words, the minimum change in value is 1. For the wheel axis, it might be larger.

Checking for Lost Mouse Input

[C++]
When you have set the cooperative level to DISCL_FOREGROUND and the focus switches to another application (or even to the menu in your own application), Microsoft® Windows® might force your application to unacquire the mouse. For this reason, you should check for the DIERR_INPUTLOST return value from the **IDirectInputDevice8::GetDeviceData** or the

IDirectInputDevice8::GetDeviceState method, and attempt to reacquire the mouse if necessary. (See Acquiring Devices.)

Note
You should not attempt to reacquire the mouse on getting a DIERR_NOTACQUIRED error. If you do, you could get caught in an infinite loop—acquisition would again fail, you would get another DIERR_NOTACQUIRED error, and so on.

[Visual Basic]
When you have set the cooperative level to DISCL_FOREGROUND and the focus switches to another application (or even to the menu in your own application), Microsoft® Windows® might force your application to unacquire the mouse. For this reason, you should check for the DIERR_INPUTLOST return value from the **DirectInputDevice8.GetDeviceData** or the **DirectInputDevice8.GetDeviceStateMouse** method, and attempt to reacquire the mouse if necessary. (See Acquiring Devices.)

Note
You should not attempt to reacquire the mouse on getting a DIERR_NOTACQUIRED error. If you do, you could get caught in an infinite loop: acquisition would again fail, you would get another DIERR_NOTACQUIRED error, and so on.

Keyboard Data

When using Microsoft® DirectInput®, consider the keyboard not as a text input device but as a game pad with many buttons. When your application requires text input, do not use DirectInput methods. It is far easier to retrieve the data from the normal Microsoft Windows® messages, which enable you take advantage of services such as character repeat and translation of physical keys to virtual keys. This is particularly important for languages other than English, which can require special translations of key presses.

[C++]
To set up the keyboard device for data retrieval, you must first call the **IDirectInputDevice8::SetDataFormat** method with the *c_dfDIKeyboard* global variable as the parameter. (See Device Data Formats.)

[Visual Basic]

To set up the keyboard device for data retrieval, you must first call the **DirectInputDevice8.SetCommonDataFormat** method with DIFORMAT_KEYBOARD as the parameter.

The following sections give more information about obtaining and interpreting keyboard data:

- Immediate Keyboard Data
- Buffered Keyboard Data
- Interpreting Keyboard Data
- Checking for Lost Keyboard Input

Immediate Keyboard Data

[C++]
To retrieve the current state of the keyboard, call the **IDirectInputDevice8::GetDeviceState** method with a pointer to an array of 256 bytes that will hold the returned data.

The **GetDeviceState** method behaves in the same way as the Microsoft® Win32® **GetKeyboardState** function. It returns a snapshot of the current state of the keyboard. Each key is represented by a byte in the array of 256 bytes whose address was passed as the *lpvData* parameter. If the high bit of the byte is set, the key is down. The array is most conveniently indexed with the Microsoft® DirectInput® Keyboard Device Constants. (See also Interpreting Keyboard Data.)

The following code example calls a function in response to the ESC key being down. Assume that *lpdiKeyboard* is an acquired DirectInput device.

```
BYTE  diKeys[256];
if (lpdiKeyboard->GetDeviceState(256, diKeys) == DI_OK)
{
  if (diKeys[DIK_ESCAPE] & 0x80) DoSomething();
}
```

[Visual Basic]
To retrieve the current state of the keyboard, call the **DirectInputDevice8.GetDeviceStateKeyboard** method, passing a **DIKEYBOARDSTATE** type.

The **GetDeviceState** method returns a snapshot of the current state of the keyboard. Each key is represented by an element in the array of 256 bytes that makes up the **DIKEYBOARDSTATE** type. If the high bit of the byte is set, the key is down. The array is most conveniently indexed with the members of the **CONST_DIKEYFLAGS** enumeration. (See also Interpreting Keyboard Data.)

The following code example determines whether the ESC key is currently being pressed. Assume that *objDIDev* is a **DirectInputDevice8** object.

```
Dim KeyState As DIKEYBOARDSTATE

Call objDIDev.GetDeviceStateKeyboard(KeyState)
If (KeyState.Key(DIK_ESCAPE) And &H80) Then
   ' Key is down.
End If
```

Buffered Keyboard Data

To retrieve buffered data from the keyboard, you must first set the buffer size (see Device Properties). This step is essential because the default size of the buffer is 0.

[C++]
You must also declare an array of **DIDEVICEOBJECTDATA** structures. This array can have up to the same number of elements as the buffer size. You do not have to retrieve the entire contents of the buffer with a single call. You can have just one element in the array and retrieve events one at a time until the buffer is empty.

After acquiring the keyboard device, you can examine and flush the buffer at any time by using the **IDirectInputDevice8::GetDeviceData** method. (See Buffered and Immediate Data.)

Each element in the **DIDEVICEOBJECTDATA** array represents a change in state for a single key—that is, a press or release. Because Microsoft® DirectInput® gets the data directly from the keyboard, any settings for character repeat in Control Panel are ignored. This means that a keystroke is counted only once, no matter how long the key is held down.

You can determine which key an element in the array refers to by checking the **dwOfs** member of the **DIDEVICEOBJECTDATA** structure against the DirectInput Keyboard Device Constants. (See also Interpreting Keyboard Data.)

If you are using Action Mapping, ignore the value in **dwOfs** and instead retrieve the application-defined data associated with the key event from the **uAppData** member.

The data for the change of state of the key is located in the **dwData** member of the **DIDEVICEOBJECTDATA** structure. Only the low byte of **dwData** is significant. The high bit of this byte is set if the key was pressed, and it is clear if it was released. In other words, the key was pressed if (**dwData** & 0x80) is nonzero.

[Visual Basic]
You must also declare an array of **DIDEVICEOBJECTDATA** types. This array can have up to the same number of elements as the buffer size. You do not have to

retrieve the entire contents of the buffer with a single call. You can have just one element in the array and retrieve events one at a time until the buffer is empty.

After acquiring the keyboard device, you can examine and flush the buffer at any time by using the **DirectInputDevice8.GetDeviceData** method. (See Buffered and Immediate Data.)

Each element in the **DIDEVICEOBJECTDATA** array represents a change in state for a single key—that is, a press or release. Because Microsoft® DirectInput® gets the data directly from the keyboard, any settings for character repeat in Control Panel are ignored. This means that a keystroke is counted only once, no matter how long the key is held down.

You can determine which key an element in the array refers to by checking the **lOfs** member of the **DIDEVICEOBJECTDATA** type against the constants in the **CONST_DIKEYFLAGS** enumeration. (See also Interpreting Keyboard Data.)

The data for the change of state of the key is located in the **lData** member of the **DIDEVICEOBJECTDATA** type. Only the low byte of **lData** is significant. The high bit of this byte is set if the key was pressed, and it is clear if it was released. In other words, the key was pressed if (**lData And** &H80) is nonzero.

Interpreting Keyboard Data

[C++]
This topic covers the identification of keys for which data is reported by the **IDirectInputDevice8::GetDeviceState** and **IDirectInputDevice8::GetDeviceData** methods. For more information on interpreting the data from **GetDeviceData**, see Time Stamps and Sequence Numbers.

[Visual Basic]
This topic covers the identification of keys for which data is reported by the **DirectInputDevice8.GetDeviceState** and **DirectInputDevice8.GetDeviceData** methods. For more information on interpreting the data from **GetDeviceData**, see Time Stamps and Sequence Numbers.

In one important respect, Microsoft® DirectInput® applications read the keyboard differently from the way Microsoft Windows® does. For DirectInput applications, keyboard data refers not to virtual keys but to the actual physical keys—that is, the scan codes. DIK_ENTER, for example, refers only to the ENTER key on the main keyboard, not to the ENTER key on the numerical keypad.

Additionally, differences among keyboards can affect the way DirectInput applications interpret keyboard data. For example, DirectInput defines a constant for each key on the enhanced keyboard, as well as the additional keys on international

keyboards. Because NEC keyboards support different scan codes than the PC-enhanced keyboards, DirectInput translates NEC key-scan codes into PC-enhanced scan codes where possible.

Other keyboard differences to consider are:

- Not all PC-enhanced keyboards have the Windows logo keys (DIK_LWIN, DIK_RWIN, and DIK_APPS), there is no way to determine whether the keys are physically available.
- Laptops and other small computers often do not implement a full set of keys. Instead, some keys (typically numeric keypad keys) are multiplexed with other keys, which are selected by an auxiliary mode key that does not generate a separate scan code.
- If the keyboard subtype indicates a PC XT or PC AT keyboard, the following keys are not available: DIK_F11, DIK_F12, and all the extended keys (DIK_* values greater than 0x7F). Furthermore, the PC XT keyboard lacks DIK_SYSRQ.

[C++]

- Japanese keyboards, particularly the NEC PC-98 keyboards, contain a substantially different set of keys than U.S. keyboards. For more information, see DirectInput and Japanese Keyboards.

[Visual Basic]

- Japanese keyboards, particularly the NEC PC-98 keyboards, contain a substantially different set of keys than U.S. keyboards. For more information, see DirectInput and Japanese Keyboards.

Checking for Lost Keyboard Input

[C++]

When you have set the cooperative level to DISCL_FOREGROUND and the focus switches to another application, Microsoft® Windows® might force your application to unacquire the keyboard. For this reason, you should check for the DIERR_INPUTLOST return value from the **IDirectInputDevice8::GetDeviceData** or **IDirectInputDevice8::GetDeviceState** methods and attempt to reacquire the keyboard, if necessary. (See Acquiring Devices.)

Note

You should not attempt to reacquire the keyboard on getting a DIERR_NOTACQUIRED error. If you do, you could get caught in an infinite loop: acquisition would again fail, you would get another DIERR_NOTACQUIRED error, and so on.

[Visual Basic]
Because Microsoft® Windows® might force your application to unacquire the keyboard when you have set the cooperative level to DISCL_FOREGROUND and the focus switches to another application, you should check for the DIERR_INPUTLOST return value from the **DirectInputDevice8.GetDeviceData** or the **DirectInputDevice8.GetDeviceStateKeyboard** method and attempt to reacquire the keyboard, if necessary. (See Acquiring Devices.)

Note

You should not attempt to reacquire the keyboard on getting a DIERR_NOTACQUIRED error. If you do, you could get caught in an infinite loop: acquisition would fail, you would get another DIERR_NOTACQUIRED error, and so on.

Joystick Data

[C++]
To set up the joystick device for data retrieval, first call the **IDirectInputDevice8::SetDataFormat** method with the *c_dfDIJoystick* or the *c_dfDIJoystick2* global variable as the parameter value. (See Device Data Formats.)

Because some device drivers do not notify Microsoft® DirectInput® of changes in state until explicitly asked to do so, you should always call the **IDirectInputDevice8::Poll** method before attempting to retrieve data from the joystick. For more information, see Polling and Event Notification.

[Visual Basic]
To set up the joystick device for data retrieval, first call the **DirectInputDevice8.SetCommonDataFormat** method with DIFORMAT_JOYSTICK or DIFORMAT_JOYSTICK2 as the parameter value. (See Device Data Formats.)

Because some device drivers do not notify Microsoft® DirectInput® of changes in state until explicitly asked to do so, you should always call the **DirectInputDevice8.Poll** method before attempting to retrieve data from the joystick. For more information, see Polling and Event Notification.

The following sections cover retrieval and interpretation of data from a joystick or other similar input device such as a game pad or steering wheel:

- Immediate Joystick Data
- Buffered Joystick Data

- Interpreting Joystick Axis Data
- Checking for Lost Joystick Input

Immediate Joystick Data

[C++]
To retrieve the current state of the joystick, call the **IDirectInputDevice8::GetDeviceState** method with a pointer to a **DIJOYSTATE** or a **DIJOYSTATE2** structure, depending on whether the data format was set with *c_dfDIJoystick* or *c_dfDIJoystick2*. (See Device Data Formats.) The joystick state returned in the structure includes the coordinates of the axes, the state of the buttons, and the state of the point-of-view controllers.

The first seven members of the **DIJOYSTATE** structure hold the axis coordinates. The last of these seven, **rglSlider**, is an array of two values. (See Interpreting Joystick Axis Data.)

The **rgdwPOV** member contains the position of up to four point-of-view controllers in hundredths of a degree clockwise from north (or forward). The center (neutral) position is reported as –1. For controllers that have only five positions, the position is one of the following values:

- –1
- 0
- 90 * DI_DEGREES
- 180 * DI_DEGREES
- 270 * DI_DEGREES.

Some drivers report a value of 65,535, instead of –1, for the center position. You should check for a centered POV indicator as follows:

```
BOOL POVCentered = (LOWORD(dwPOV) == 0xFFFF);
```

The **rgbButtons** member is an array of bytes, one for each of 32 or 128 buttons, depending on the data format. For each button, the high bit is set if the button is down and it is clear if the button is up or not present.

The **DIJOYSTATE2** structure has additional members for information about the velocity, acceleration, force, and torque of the axes.

[Visual Basic]
To retrieve the current state of the joystick, call the **DirectInputDevice8.GetDeviceStateJoystick** or the **DirectInputDevice8.GetDeviceStateJoystick2** method, depending on whether the data format was set with DIFORMAT_JOYSTICK or DIFORMAT_JOYSTICK2. (See Device Data Formats.) The joystick state returned in the *state* parameter includes

the coordinates of the axes, the state of the buttons, and the state of the point-of-view controllers.

The **POV** member of the **DIJOYSTATE** or the **DIJOYSTATE2** type contains the position of up to four point-of-view controllers in hundredths of a degree clockwise from north (or forward). The center position is reported as –1. For controllers that have only five positions, the position is one of the following values:

- –1
- 0
- 9000
- 18000
- 27000

Some drivers report a value of 65,535, instead of –1, for the neutral position. You should check for a centered POV indicator as follows:

```
Dim POVCentered As Boolean
POVCentered = MyDijoystate.POV(0) And &HFFFF
```

The **buttons** member is an array of bytes, one for each of 32 or 128 buttons, depending on the data type. For each button, the high bit is set if the button is down, and it is clear if the button is up or not present.

The **DIJOYSTATE2** type has additional members for information about the velocity, acceleration, force, and torque of the axes.

For more information, see Interpreting Joystick Axis Data.

See also

Buffered and Immediate Data

Buffered Joystick Data

[C++]
To retrieve buffered data from the joystick, first set the buffer size (see Device Properties) and declare an array of **DIDEVICEOBJECTDATA** structures. This array can have up to the same number of elements as the buffer size. You do not have to retrieve the entire contents of the buffer with a single call. You can have just one element in the array and retrieve events one at a time until the buffer is empty.

After acquiring the device, you can examine and flush the buffer at any time with the **IDirectInputDevice8::GetDeviceData** method. (See Buffered and Immediate Data.)

Each element in the **DIDEVICEOBJECTDATA** array represents a change in state for a single object on the joystick. For instance, if the user presses button 0 and moves the stick diagonally, the array passed to **GetDeviceData** (if it has at least three

elements, and *pdwInOut* is at least 3) will have three elements filled in—an element for button 0 being pressed, an element for the change in the x-axis, and an element for the change in the y-axis—and the value of *pdwInOut* will be set to 3.

You can determine which object an element in the array refers to by checking the **dwOfs** member of the **DIDEVICEOBJECTDATA** structure against the following values:

- DIJOFS_X
- DIJOFS_Y
- DIJOFS_Z
- DIJOFS_Rx
- DIJOFS_Ry
- DIJOFS_Rz
- DIJOFS_BUTTON0 *to* DIJOFS_BUTTON31 *or* DIJOFS_BUTTON(*n*)
- DIJOFS_POV(*n*)
- DIJOFS_SLIDER(*n*)

Each of these values is equivalent to the offset of the data for the object in a **DIJOYSTATE** structure. For example, DIJOFS_BUTTON0 is equivalent to the offset of **rgbButtons[0]** in the **DIJOYSTATE** structure. You can use simple comparisons to determine which device object is associated with an item in the buffer. For example:

```
DIDEVICEOBJECTDATA  *lpdidod;
int             n;
.
.
.
/* JoyBuffer is an array of DIDEVICEOBJECTDATA structures
   that has been set by a call to GetDeviceData.
   n is incremented in a loop that examines all filled elements
   in the array. */
lpdidod = &JoyBuffer[n];
if (( (int) lpdidod->dwOfs == DIJOFS_BUTTON0)
   && (lpdidod->dwData & 0x80))
{
 ;  // Do something in response to press of primary button.
}
```

If the data format was set with *c_dfDIJoystick2*, you can use the predefined offsets for all the device objects that exist in **DIJOYSTATE**, but you must supply your own offsets for device objects represented in the extra members of **DIJOYSTATE2**.

If you are using Action Mapping, ignore the value in **dwOfs** and instead retrieve the application-defined data associated with the event from the **uAppData** member.

The data for the change of state of the device object is located in the **dwData** member of the **DIDEVICEOBJECTDATA** structure. For axes, the coordinate value is returned in this member. For button objects, only the low byte of **dwData** is significant. The high bit of this byte is set if the button is pressed, and it is clear if the button is released.

For the other members, see Time Stamps and Sequence Numbers.

[Visual Basic]
To retrieve buffered data from the joystick, first set the buffer size (see Device Properties) and declare an array of **DIDEVICEOBJECTDATA** types. This array can have up to the same number of elements as the buffer size. You do not have to retrieve the entire contents of the buffer with a single call. You can have just one element in the array and retrieve events one at a time until the buffer is empty.

After acquiring the device, you can examine and flush the buffer at any time with the **DirectInputDevice8.GetDeviceData** method. (See Buffered and Immediate Data.)

Each element in the **DIDEVICEOBJECTDATA** array represents a change in state for a single object on the joystick. For instance, if the user presses button 0 and moves the stick diagonally, the array passed to **GetDeviceData** (if it has at least three elements) will have three elements filled in—an element for button 0 being pressed, an element for the change in the x-axis, and an element for the change in the y-axis—and the return value of the method will be 3.

You can determine which object an element in the array refers to by checking the **lOfs** member of the **DIDEVICEOBJECTDATA** type against the constants of the **CONST_DIJOYSTICKOFS** enumeration. Each of these values is equivalent to the offset of the data for the object in a **DIJOYSTATE** type. For example, DIJOFS_BUTTON0 is equivalent to the offset of **buttons(0)** in the **DIJOYSTATE** type.

If the data format was set with DIFORMAT_JOYSTICK2, you can use the offset constants for all the device objects that exist in **DIJOYSTATE**, but you must supply your own offsets for device objects represented in the extra members of **DIJOYSTATE2**. The internal organization of this type is the same as that of the equivalent C++ structure in the Dinput.h header file.

The data for the change of state of the device object is located in the **lData** member of the **DIDEVICEOBJECTDATA** type. For axes, the coordinate value is returned in this member. For button objects, only the low byte of **lData** is significant. The high bit of this byte is set if the button is pressed, and it is clear if the button is released.

For the other members, see Time Stamps and Sequence Numbers.

Interpreting Joystick Axis Data

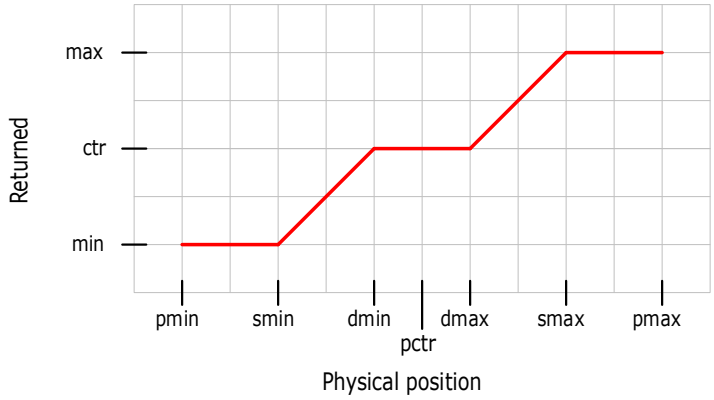
Axis values for the joystick are like those for the mouse. The value returned for the x-axis is greater as the stick moves to the right, and the value for the y-axis increases as the stick moves toward the user.

Data is in arbitrary units determined by the range property of the axis. For example, if the range for the stick's x-axis is from 0 through 10,000, a unit is one ten-thousandth of the stick's left-right travel, and the center position is 5,000. For some axes, the granularity property might be greater than 1. In this case, values are rounded off. For example, if the granularity is 10, values are reported as 0, 10, 20, and so on.

Axis data is also affected by the dead zone, a region around the center position in which motion is ignored. The dead zone provides tolerance for a slight deviation from the true center position for either or both axes of the stick. An axis value within the range of the dead zone is reported as true center.

The saturation property of an axis is a zone of tolerance at the minimum and maximum of the range. An axis value within this zone is reported as the minimum or maximum value. The purpose of the saturation property is to allow for slight differences between, for example, the minimum x-axis value reported at the top-left and bottom-left positions of the stick.

The following illustration shows the effect of the dead zone and the saturation zones. The vertical axis represents the returned axis values, where *min* and *max* are the lower and upper limits of the reported range and *ctr* is the reported center. The horizontal axis shows the physical position of the stick, where *pmin* and *pmax* are the extremes of the physical range, *pctr* is neutral position of the axis, *dmin* and *dmax* are the limits of the dead zone, and *smin* and *smax* are the boundaries of the lower and upper saturation zones. The lower saturation zone lies between *pmin* and *smin*, the upper saturation zone lies between *smax* and *pmax*, and the dead zone lies between *dmin* and *dmax*.



[C++]

For more information on joystick properties, see the following:

- **IDirectInputDevice8::GetProperty**
- **IDirectInputDevice8::SetProperty**
- **DIPROPRANGE**
- **DIACTIONFORMAT**

[Visual Basic]
For more information on joystick properties, see the following:

- **DirectInputDevice8.GetProperty**
- **DirectInputDevice8.SetProperty**
- **DIPROPRANGE**
- **DIACTIONFORMAT**

Axis coordinates from the joystick can be either relative or absolute. (See Relative and Absolute Axis Coordinates.) Because a joystick is an absolute device—unlike a mouse, it cannot travel infinitely far along any axis—absolute data is returned by default. When the axis mode for the joystick is set to relative, the axis coordinate represents the number of units of movement along the axis since the last value was returned.

[C++]
The axis mode, which specifies whether relative or absolute data should be returned, is a property that can be changed before the device is acquired. (See Device Properties.) To set the axis mode to relative, call the **IDirectInputDevice8::SetProperty** method with the DIPROP_AXISMODE value in the *rguidProp* parameter and with DIPROPAXISMODE_REL in the **dwData** member of the **DIPROPDWORD** structure.

[Visual Basic]
The axis mode, which specifies whether relative or absolute data should be returned, is a property that can be changed before the device is acquired. (See Device Properties.) To set the axis mode to relative, call **DirectInputDevice8.SetProperty** with "DIPROP_AXISMODE" in the *guid* parameter and with DIPROPAXISMODE_REL in the **lData** member of the **DIPROPLONG** type.

Using CPOINTs

[C++]
The **CPOINT** structure is used to calibrate segments of the response curve of a device axis. Up to eight CPOINTs can be set through the **DIPROPCPOINTS** structure. Each **CPOINT** structure contains a raw data value representing the position of the joystick along the axis, and a logical value. When your application receives the raw value, the logical value is used to calculate the returned value that is actually used. Raw values that fall between specific CPOINT raw data values are interpolated to logical values based on a line drawn between the CPOINTs to either side of the value.

The following is an example of the effect of five CPOINTs applied to the response curve of a joystick's X axis. Before application of the **DIPROPCPOINTS** structure, the response graph for the example axis would appear as shown below, including saturation and dead zones.

A **DIPROPCPOINTS** structure is initialized as follows, assuming the physical range to be -500 to 500.

```
DIPROPCPOINTS dipcp;

dipcp.diph.dwSize = sizeof(DIPROPCPOINTS);
dipcp.diph.dwHeaderSize = sizeof(DIPROPHEADER);
dipcp.diph.dwObj = GUID_XAxis;
dipcp.diph.dwHow = DIPH_BYID;
dipcp.dwCPointsNum = 5;
dipcp.cp[0].lP = -500
dipcp.cp[0].dwLog = 0
dipcp.cp[1].lP = -300
dipcp.cp[1].dwLog = 1000
dipcp.cp[2].lP = -100
dipcp.cp[2].dwLog = 5000
dipcp.cp[3].lP = 200
dipcp.cp[3].dwLog = 9000
dipcp.cp[4].lP = 500
dipcp.cp[4].dwLog = 10000
```

Note that you do not need to know the actual maximum returned value as the values in the dwLog member of the **CPOINT** structure are expressed as a percentage of the total, multiplied by 10000 (the number of discrete divisions in the total range). For instance, the example sets the curve so that 10% of the maximum value is returned at the -300 point in the joystick's range. 0.10 * 10000 = 1000, so that value is given to the *dwLog* member while -300 is assigned to the corresponding *lP* member.

After applying this **DIPROPCPOINTS** structure through **IDirectInputDevice8::SetProperty**, the curve responds as seen below.

[Visual Basic]
This topic does not apply to Microsoft® Visual Basic®.

Checking for Lost Joystick Input

If your application is using the joystick in foreground mode (see Cooperative Levels), it will lose the device when the focus shifts to another application.

[C++]
Check for the DIERR_INPUTLOST return value from the **IDirectInputDevice8::GetDeviceData** or the **IDirectInputDevice8::GetDeviceState** method and attempt to reacquire the joystick, if necessary. (See Acquiring Devices.)

Note

You should not attempt to reacquire the joystick on getting a DIERR_NOTACQUIRED error. If you do, you could get caught in an infinite loop: acquisition would fail again, you would get another DIERR_NOTACQUIRED error, and so on.

[Visual Basic]
Check for the DIERR_INPUTLOST return value from the **DirectInputDevice8.GetDeviceData** or the **DirectInputDevice8.GetDeviceStateKeyboard** method and attempt to reacquire the keyboard, if necessary. (See Acquiring Devices.)

Note

You should not attempt to reacquire the keyboard on getting a DIERR_NOTACQUIRED error. If you do, you could get caught in an infinite loop: acquisition would fail, you would get another DIERR_NOTACQUIRED error, and so on.

Because access to the joystick is not lost except when your application moves to the background—unlike the mouse and keyboard, the joystick is never used by the Microsoft® Windows® system—an alternative method is to reacquire the device in response to a WM_ACTIVATE message.

Output Data

[C++]

Some Human Interface Devices both generate input and accept output. The **IDirectInputDevice8::SendDeviceData** method is used to send packets of data to such devices.

SendDeviceData may be viewed as **IDirectInputDevice8::GetDeviceData** in reverse. As with that method, it uses the **DIDEVICEOBJECTDATA** structure as the basic unit of data. In this case, however, the **dwOfs** member contains the instance ID of the device object associated with the data rather than its offset in the data format for the device. (Because offset identifiers exist only for device objects that provide input in the selected data format, an object that accepts only output might not even have an offset.) The **dwData** member contains whatever data is appropriate for the object. The **dwTimeStamp** and **dwSequence** members are not used and must be set to 0.

To send data to the device, set up an array of **DIDEVICEOBJECTDATA** structures, fill the required number of elements with data, and then pass the array address and the number of elements used to **SendDeviceData**. Data for different device objects is combined into a single packet that is then sent to the device.

The form of the data packet is specific to the device, as is the treatment of unused fields in the packet. Some devices may treat fields as optional. In other words, if no data is supplied, the state of the object remains unchanged. More commonly, all fields are significant, even when you do not specifically supply data for them. For example, if you send data to a single keyboard LED, it is assumed that the data for the other two LEDs is 0 and that they are turned off. However, you can override this behavior by using the DISDD_CONTINUE flag. In this case, the data for the other two LEDs is the value you most recently sent them.

The device object type identifiers are obtained from the **dwType** member of the **DIDEVICEOBJECTINSTANCE** structure after passing the HID usage page and usage code to **IDirectInputDevice8::GetObjectInfo**. For example, the type identifier of the Scroll Lock LED can be obtained by using the following code, where *pdev* is an **IDirectInputDevice8** interface pointer representing the keyboard.

```
DIDEVICEOBJECTINSTANCE didoi;
DWORD NumLockID;

HRESULT hr = pdev->GetObjectInfo(&didoi,
     DIMAKEUSAGEDWORD(0x07,0x53), DIPH_BYUSAGE);
NumLockID = didoi.dwType;
```

The following sample function, when called repeatedly, causes the LEDs on the keyboard to flash in a recurring pattern. It is assumed that the high bit of the data byte determines the state of the LED. The **DWORD**s *NumLockID*, *CapsLockID*, and *ScrollLockID* have all been obtained as shown above.

```
#define ARRAYSIZE 4
void FlashLEDs(void)
  {
  static int       rgiBits[] = { 1, 2, 4, 2 };
```

```
static int        iLooper = 0;
DWORD             cdod = 3;               // Number of items
DIDEVICEOBJECTDATA rgdod[ARRAYSIZE-1];
HRESULT           hres;

// Must clear dwTimeStamp and dwSequence
ZeroMemory(rgdod, sizeof(rgdod));

rgdod[0].dwOfs = NumLockID;
rgdod[1].dwOfs = CapsLockID;
rgdod[2].dwOfs = ScrollLockID;

rgdod[0].dwData = (rgiBits[iLooper] & 1) ? 0x80 : 0;
                              // NumLock light on/off
rgdod[1].dwData = (rgiBits[iLooper] & 2) ? 0x80 : 0;
                              // CapsLock light on/off
rgdod[2].dwData = (rgiBits[iLooper] & 4) ? 0x80 : 0;
                              // ScrollLock light on/off

iLooper = (iLooper + 1) % ARRAYSIZE; // Loops from 0 to 3

hres = pdev->SendDeviceData(sizeof(DIDEVICEOBJECTDATA),
      rgdod, &cdod, 0);
}
```

[Visual Basic]

Some Human Interface Devices both generate input and accept output. The **DirectInputDevice8.SendDeviceData** method is used to send packets of data to such devices.

SendDeviceData can be viewed as **DirectInputDevice8.GetDeviceData** in reverse. As with that method, it uses the **DIDEVICEOBJECTDATA** type as the basic unit of data. In this case, however, the **lOfs** member contains the instance ID of the device object associated with the data, rather than its offset in the data format for the device. This ID can be extracted from the value returned by **DirectInputDeviceObjectInstance.GetType** after device objects have been enumerated. (Because offset identifiers exist only for device objects that provide input in the selected data format, an object that accepts only output might not even have an offset.) The **lData** member contains whatever data is appropriate for the object. The **lTimeStamp** and **lSequence** members are not used and must be set to 0.

To send data to the device, first set up an array of **DIDEVICEOBJECTDATA** types, fill the required number of elements with data, and then pass its address and the number of elements used to **SendDeviceData**. Data for different device objects is combined into a single packet that is then sent to the device.

The form of the data packet is specific to the device, as is the treatment of unused fields in the packet. Some devices treat fields as optional. In other words, if no data is supplied, the state of the object remains unchanged. More commonly, all fields are significant, even when you do not specifically supply data for them. For example, if you send data to a single keyboard LED, it is assumed that the data for the other two LEDs is 0 and that they are turned off. However, you can override this behavior by using the DISDD_CONTINUE flag. In this case, the data for the other two LEDs is the value you most recently sent them.

The device object type identifiers are obtained after passing the HID usage page and usage code to **DirectInputDevice8.GetObjectInfo**.

The following sample function, when called repeatedly, causes the LEDs on the keyboard to flash in a recurring pattern. It is assumed that the high bit of the data byte determines the state of the LED. The device object instance identifiers *NumLockID*, *CapsLockID*, and *ScrollLockID* have all been previously obtained, and *diDevKeyboard* is a previously defined DirectInputDevice8 object.

```
Private Sub FlashLEDs()
    Const ARRAYSIZE = 4
    Static rgiBits(ARRAYSIZE-1) As Integer
      rgiBits(0) = 1
      rgiBits(1) = 2
      rgiBits(2) = 4
      rgiBits(3) = 2
    Static iLooper As Integer
      iLooper = 0
    Dim cdod As Long
      cdod = 3              ' Number of items
    Dim rgdod(3) As DIDEVICEOBJECTDATA
    Dim i as Integer

    ' lTimeStamp and lSequence must be set to 0
    For i = 0 To 3
       rgdod(i).lTimeStamp = 0
       rgdod(i).lSequence = 0
    Next i

    rgdod(0).lOfs = NumLockID
    rgdod(1).lOfs = CapsLockID
    rgdod(2).lOfs = ScrollLockID

    Select Case (rgiBits(iLooper) And 1)
       Case Is = 0
          rgdod(0).lData = 0
       Case Is = 1
          rgdod(0).lData = &H80         ' NumLock light on/off
    End Select
```

```
    Select Case (rgiBits(iLooper) And 2)
        Case Is = 0
            rgdod(1).lData = 0
        Case Is = 1
            rgdod(1).lData = &H80          ' CapsLock light on/off
    End Select

    Select Case (rgiBits(iLooper) And 4)
        Case Is = 0
            rgdod(2).lData = 0
        Case Is = 1
            rgdod(2).lData = &H80          ' ScrollLock light on/off
    End Select

    iLooper = (iLooper + 1) Mod ARRAYSIZE  ' Loops from 0 to 3

    Call diDevKeyboard.SendDeviceData(cdod, rgdod, 0)

End Sub
```

Action Mapping

Traditionally, applications have done their own mapping of events to particular buttons and axes. A car-racing game, for example, might assume that the x-axis on the user's joystick or mouse was the most suitable control for steering the car. The only way to accommodate new or unusual devices was to provide configuration options so that the user could specify some other axis, such as a rotational axis, to use for steering. Moreover, the application had no way of knowing which installed device was the best fit for the game, so the user typically had to choose a device from a menu or make sure only the preferred device was attached.

Using action mapping, you no longer need to make assumptions about the best use of devices and device objects. Instead, your application binds actions to virtual controls wherever possible. Rather than getting data from the x-axis and steering the car to the left or the right accordingly, the application might get data from a virtual control called DIAXIS_DRIVINGR_STEER. Microsoft® DirectInput® assigns the virtual control to a physical control—that is, a device object. It does so by taking into account the application genre, user preferences, information from the device manufacturer, and the user's configuration of the device.

Action mapping also simplifies the input loop by returning data for all devices in a form independent of the particular device. A single action can be mapped to more

than one device, and the input loop can respond to the action the same way regardless of which device is being read.

The following topics contain more information on the steps required to implement action mapping.

- Preparing the Action Map
- Finding Matching Devices
- Configuring the Action Map
- User Configuration of the Device
- Retrieving Action Data
- Maintaining Files During Development

Preparing the Action Map

[C++]
The action map is a **DIACTIONFORMAT** structure containing information about application actions and their mapping to virtual controls or device objects. The structure is passed back and forth between the application and Microsoft® DirectInput® to establish the final mapping. This section explains how to initialize the map.

1. Define Application Actions

The first step in implementing DirectInput action mapping is to determine what input-driven actions in your application need to be mapped to device objects. For actions that can be performed either by an axis or by a button, you must define separate actions for both input types. It is recommended that you define button actions for all important functions, in case the device does not have the appropriate axes.

The following sample enumeration of action values might be defined by a car-racing game. Axis actions begin with "eA" and button actions with "eB".

```
enum eGameActions
{
    eA_STEER,       // Steering
    eB_STEER_LEFT,  // Steer left
    eB_STEER_RIGHT, // Steer right
    eA_ACCELERATE,  // Change speed
    eB_ACCELERATE,  // Speed up
    eB_DECELERATE,  // Slow down
    eA_BRAKE,       // Brake
    eB_BRAKE,       // Brake
    eB_UPSHIFT,     // Shift to higher gear
    eB_DOWNSHIFT,   // Shift to lower gear
    eB_CYCLEVIEW,   // Cycle to next view
```

```
    eB_COURSEVIEW,  // Toggle course view
    eB_DRIVERVIEW,  // View from driver's seat
    eB_BRAKEBIAS,   // Brake bias
    eA_VOLUME,      // Sound volume
    eB_MUTE         // Toggle sound
};

#define NUM_MY_ACTIONS 16
```

In the example, actions are defined as enumerated values. However, they could be other 32-bit data types, such as pointers to functions. When you retrieve device data, you get whatever action value you have defined, and you can handle it in any way you like.

[Visual Basic]
The action map is a **DIACTIONFORMAT** type containing information about application actions and their mapping to virtual controls or device objects. The type is passed back and forth between the application and Microsoft® DirectInput® to establish the final mapping. This section explains how to initialize the map.

1. Define Application Actions

The first step in implementing DirectInput action mapping is to determine what input-driven actions in your application need to be mapped to device objects. For actions that can be performed either by an axis or by a button, you must define separate actions for both input types. It is recommended that you define button actions for all important functions, in case the device does not have the appropriate axes.

The following sample enumeration of action values might be defined by a car-racing game. Axis actions begin with "eA" and button actions with "eB".

```
Enum eGameActions
    eA_STEER        'Steering
    eB_STEER_LEFT   'Steer left
    eB_STEER_RIGHT  'Steer right
    eA_ACCELERATE   'Change speed
    eB_ACCELERATE   'Speed up
    eB_DECELERATE   'Slow down
    eA_BRAKE        'Brake
    eB_BRAKE        'Brake
    eB_UPSHIFT      'Shift to higher gear
    eB_DOWNSHIFT    'Shift to lower gear
    eB_CYCLEVIEW    'Cycle to next view
    eB_COURSEVIEW   'Toggle course view
    eB_DRIVERVIEW   'View from driver's seat
    eB_BRAKEBIAS    'Brake bias
```

```
    eA_VOLUME       'Sound volume
    eB_MUTE         'Toggle sound
End Enum

Const NUM_MY_ACTIONS = 16
```

In the example, actions are defined as enumerated values. However, they could be other 32-bit data types. When you retrieve device data, you get whatever action value you have defined, and you can handle it in any way you like.

[C++]

2. Define the Genre

The next step is to decide what genre your application belongs to. A genre defines a set of virtual controls. By selecting the proper genre, you can obtain the best possible fit of virtual controls to application actions. Manufacturers who choose to supply default mappings for their devices must support one or more of the genres defined by DirectInput. See Action Mapping Constants for a list of these genres.

[Visual Basic]

2. Define the Genre

The next step is to decide what genre your application belongs to. A genre defines a set of virtual controls. By selecting the proper genre, you can obtain the best possible fit of virtual controls to application actions. Manufacturers who choose to supply default mappings for their devices must support one or more of the genres defined by DirectInput. See Action Mapping Constants for a list of these genres.

For the game in the example, the obvious choice is the DIVIRTUAL_DRIVING_RACE genre, which contains the following virtual controls.

Priority 1 Controls

DIAXIS_DRIVINGR_STEER
DIAXIS_DRIVINGR_ACCELERATE
DIAXIS_DRIVINGR_BRAKE
DIBUTTON_DRIVINGR_SHIFTUP
DIBUTTON_DRIVINGR_SHIFTDOWN
DIBUTTON_DRIVINGR_VIEW
DIBUTTON_DRIVINGR_MENU

Priority 2 Controls

DIAXIS_DRIVINGR_ACCEL_AND_BRAKE
DIHATSWITCH_DRIVINGR_GLANCE
DIBUTTON_DRIVINGR_ACCELERATE_LINK
DIBUTTON_DRIVINGR_AIDS
DIBUTTON_DRIVINGR_BOOST
DIBUTTON_DRIVINGR_BRAKE
DIBUTTON_DRIVINGR_DASHBOARD
DIBUTTON_DRIVINGR_DEVICE
DIBUTTON_DRIVINGR_GLANCE_LEFT_LINK
DIBUTTON_DRIVINGR_GLANCE_RIGHT_LINK
DIBUTTON_DRIVINGR_MAP
DIBUTTON_DRIVINGR_PAUSE
DIBUTTON_DRIVINGR_PIT
DIBUTTON_DRIVINGR_STEER_LEFT_LINK
DIBUTTON_DRIVINGR_STEER_RIGHT_LINK

There is no difference in functionality between Priority 1 and Priority 2 controls. Priority 1 controls are those most likely to be supported by device manufacturers in their default mappings. However, there is no guarantee that any virtual control will be supported by a device.

[C++]

3. Assign Actions to Controls or Device Objects

The next step in creating the action map is to associate each application action with one or more of the virtual controls defined for the genre. You do this by declaring and initializing an array of **DIACTION** structures. Each structure in the array specifies the action value, the virtual control to associate with it, and a friendly name that describes the action. Leave other members as zero; they will be filled in later by DirectInput.

You can also use elements of the **DIACTION** array to map actions to particular keys or buttons on the keyboard or mouse or to channels on a Microsoft DirectPlay® voice device. By doing so, you can take advantage of the simplified input loop for all input, not just that from virtual controls. For example, suppose you map the application-defined action eB_UPSHIFT to both the DIBUTTON_DRIVINGR_SHIFTUP virtual control and to the Page Up key. When retrieving data, you get back eB_UPSHIFT whether the input came from a joystick button or the keyboard.

The following example declares an action map for the car-racing game.

```
DIACTION rgActions[]=
{
//Genre-defined virtual axes

  {eA_STEER,      DIAXIS_DRIVINGR_STEER,       0, "Steer",      },
  {eA_ACCELERATE, DIAXIS_DRIVINGR_ACCELERATE,  0, "Accelerate", },
  {eA_BRAKE,      DIAXIS_DRIVINGR_BRAKE,       0, "Brake",      },
```

```
//Genre-defined virtual buttons

  {eB_UPSHIFT,     DIBUTTON_DRIVINGR_SHIFTUP,    0, "Upshift",    },
  {eB_DOWNSHIFT,   DIBUTTON_DRIVINGR_SHIFTDOWN,  0, "DownShift",  },
  {eB_CYCLEVIEW,   DIBUTTON_DRIVINGR_VIEW,       0, "Change View",},

// Actions not defined in the genre that can be assigned to any
// button or axis

  {eA_VOLUME,      DIAXIS_ANY_1,                 0, "Volume",     },
  {eB_MUTE,        DIBUTTON_ANY(0),              0, "Toggle Sound",},

// Actions not defined in the genre that must be assigned to
// particular keys

  {eB_DRIVERVIEW,  DIKEYBOARD_1,                 0, "Driver View",},
  {eB_COURSEVIEW,  DIKEYBOARD_C,                 0, "Course View",},
  {eB_BRAKEBIAS,   DIKEYBOARD_B,                 0, "Brake Bias", },

// Actions mapped to keys as well as to virtual controls

  {eB_UPSHIFT,     DIKEYBOARD_PRIOR,             0, "Upshift",    },
  {eB_DOWNSHIFT,   DIKEYBOARD_NEXT,              0, "Downshift",  },
  {eB_STEER_LEFT,  DIKEYBOARD_LEFT,              0, "Steer Left", },
  {eB_STEER_RIGHT, DIKEYBOARD_RIGHT,             0, "Steer Right",},
  {eB_ACCELERATE,  DIKEYBOARD_UP,                0, "Accelerate", },
  {eB_DECELERATE,  DIKEYBOARD_DOWN,              0, "Decelerate", },
  {eB_BRAKE,       DIKEYBOARD_END,               0, "Brake",      },

// Actions mapped to buttons as well as to virtual controls and keys

  {eB_UPSHIFT,     DIMOUSE_BUTTON0,              0, "Upshift",    },
  {eB_DOWNSHIFT,   DIMOUSE_BUTTON1,              0, "Downshift",  },
};
```

In the example, some actions are mapped to actual keys by using Keyboard Mapping Constants. Similar mappings to the mouse buttons and axes can be made by using Mouse Mapping Constants.

The **DIACTION** array is contained within a **DIACTIONFORMAT** structure that also contains information about the genre, the application, and the desired scaling of axis data. Use the same instance of this structure throughout the action mapping process. Some members will not be used immediately, but you can fill in the entire structure before the next step, Finding Matching Devices.

[Visual Basic]

3. Assign Actions to Controls or Device Objects

The next step in creating the action map is to associate each application action with one or more of the virtual controls defined for the genre. You do this by declaring and initializing an array of **DIACTION** types. Each type in the array specifies the action value, the virtual control to associate with it, and a friendly name that describes the action. Leave other members as zero; they will be filled in later by DirectInput.

You can also use elements of the **DIACTION** array to map actions to particular keys or buttons on the keyboard or mouse or to channels on a Microsoft DirectPlay® voice device. By doing so, you can take advantage of the simplified input loop for all input, not just that from virtual controls. For example, suppose you map the application-defined action eB_UPSHIFT to both the DIBUTTON_DRIVINGR_SHIFTUP virtual control and to the Page Up key. When retrieving data, you get back eB_UPSHIFT whether the input came from a joystick button or the keyboard.

The following example begins the declaration of an action map for the car-racing game.

```
Dim rgActions(20) As DIACTION

' Genre-defined virtual axes
With rgActions(0)
  .lAppData = eA_STEER
  .lSemantic = DIAXIS_DRIVINGR_STEER
  .lFlags = 0
  .ActionName = "Steer"
End With
With rgActions(1)
  .lAppData = eA_ACCELERATE
  .lSemantic = DIAXIS_DRIVINGR_ACCELERATE
  .lFlags = 0
  .ActionName = "Accelerate"
End With
With rgActions(2)
  .lAppData = eA_BRAKE
  .lSemantic = DIAXIS_DRIVINGR_BRAKE
  .lFlags = 0
  .ActionName = "Brake"
End With
'and so on.
```

It may be more efficient, however, to use a Sub procedure to assign the values to the DIACTION array in the DIACTIONFORMAT type directly as in the following example.

```
Dim m_diaf As DIACTIONFORMAT
```

```
Dim m_NumberofSemantics As Long


Private Sub Form_Load()

m_NumberofSemantics = 0

AddAction eA_STEER, DIAXIS_DRIVINGR_STEER, 0, "Steer"
AddAction eA_ACCELERATE, DIAXIS_DRIVINGR_ACCELERATE, 0, "Accelerate"
AddAction eA_BRAKE, DIAXIS_DRIVINGR_BRAKE, 0, "Brake"

' Genre-defined virtual buttons

AddAction eB_UPSHIFT, DIBUTTON_DRIVINGR_SHIFTUP, 0, "Upshift"
AddAction eB_DOWNSHIFT, DIBUTTON_DRIVINGR_SHIFTDOWN, 0, "DownShift"
AddAction eB_CYCLEVIEW, DIBUTTON_DRIVINGR_VIEW, 0, "Change View"

'  Actions not defined in the genre that can be assigned to any
'  button or axis

AddAction eA_VOLUME, DIAXIS_ANY_1, 0, "Volume"
AddAction eB_MUTE, DIBUTTON_ANY, 0, "Toggle Sound"

'  Actions not defined in the genre that must be assigned to
'  particular keys

AddAction eB_DRIVERVIEW, DIKEYBOARD_1, 0, "Driver View"
AddAction eB_COURSEVIEW, DIKEYBOARD_C, 0, "Course View"
AddAction eB_BRAKEBIAS, DIKEYBOARD_B, 0, "Brake Bias"

'  Actions mapped to keys as well as to virtual controls

AddAction eB_UPSHIFT, DIKEYBOARD_PRIOR, 0, "Upshift"
AddAction eB_DOWNSHIFT, DIKEYBOARD_NEXT, 0, "Downshift"
AddAction eB_STEER_LEFT, DIKEYBOARD_LEFT, 0, "Steer Left"
AddAction eB_STEER_RIGHT, DIKEYBOARD_RIGHT, 0, "Steer Right"
AddAction eB_ACCELERATE, DIKEYBOARD_UP, 0, "Accelerate"
AddAction eB_DECELERATE, DIKEYBOARD_DOWN, 0, "Decelerate"
AddAction eB_BRAKE, DIKEYBOARD_END, 0, "Brake"

'  Actions mapped to buttons as well as to virtual controls and keys

AddAction eB_UPSHIFT, DIMOUSE_BUTTON0, 0, "Upshift"
AddAction eB_DOWNSHIFT, DIMOUSE_BUTTON1, 0, "Downshift"

End Sub
```

```
Private Sub AddAction(user As Long, semantic As Long,_
                      flags As Long, strName As String)
ReDim Preserve m_diaf.ActionArray(m_NumberofSemantics)

With m_diaf.ActionArray(m_NumberofSemantics)
   .lAppData = user
   .lSemantic = semantic
   .lFlags = flags
   .ActionName = strName
End With
m_NumberofSemantics = m_NumberofSemantics + 1

End Sub
```

In the example, some actions are mapped to actual keys by using Keyboard Mapping Constants. Similar mappings to the mouse buttons and axes can be made by using Mouse Mapping Constants.

The **DIACTION** array is contained within the **DIACTIONFORMAT** type that also contains information about the genre, the application, and the desired scaling of axis data. Use the same instance of this type throughout the action mapping process. Some members will not be used immediately, but you can fill in the entire type before the next step, Finding Matching Devices.

Finding Matching Devices

After you define the application actions and the virtual controls or device objects to which these actions are to be mapped, the next step is to enumerate devices on the system to find those that best support the desired virtual controls.

[C++]
To do so, pass the **DIACTIONFORMAT** structure to **IDirectInput8::EnumDevicesBySemantics**. This method works in much the same way as **IDirectInput8::EnumDevices** and takes a similar callback function.

[Visual Basic]
To do so, pass the **DIACTIONFORMAT** type to **DirectInput8.GetDevicesBySemantics**. This method works in much the same way as **DirectInput8.GetDIDevices**.

Devices that have been configured by the user to match certain controls are always enumerated first. For example, if a user has configured a wheel as the primary steering device for driving games, then the wheel is enumerated first whenever devices that support DIAXIS_DRIVINGR_STEER are requested, taking precedence over other capable devices such as joysticks that have not been configured by the user. Otherwise, the order in which available devices are enumerated is determined by the degree to which they match the requested controls.

[C++]
In the enumeration callback, you can retrieve the default action mapping for each device, change any mappings you don't like, give the user an opportunity to reconfigure the device, and apply the action map. These steps are covered in Configuring the Action Map. Flags returned in the **DIEnumDevicesBySemanticsCallback** will provide information about why a particular device was enumerated. These flags will indicate whether a device has been used recently, is newly installed, or will accept mappings of priority 1 or priority 2 controls.

[Visual Basic]
Once you have the enumeration, you can retrieve the default action mapping for each device, change any mappings you don't like, give the user an opportunity to reconfigure the device, and apply the action map. These steps are covered in Configuring the Action Map.

Configuring the Action Map

[C++]
As each device is enumerated, you can obtain a pointer to it, retrieve the default action map, make changes in the default map, and apply the final mappings.

1. Obtaining the Device

Obtain the **IDirectInputDevice8** interface pointer for each enumerated device from the *lpdid* parameter of the enumeration callback. See **DIEnumDevicesBySemanticsCallback**. If you want to save the device interface for use in your application, call **AddRef** on the pointer and assign it to a global variable.

[Visual Basic]
As each device is enumerated, you can retrieve the default action map, make changes in the default map, and apply the final mappings.

1. Obtaining the Device

Once the enumeration is complete, you can use the **DirectInputEnumDevices8.GetItem** method to obtain an instance of the **DirectInputDeviceInstance8** class. **DirectInputDeviceInstance8.GetGuidInstance** enables the retrieval of the device GUID, which can in turn be passed to **DirectInput8.CreateDevice**. The resulting **DirectInputDevice8** object can then be saved to a global variable for use in your application, and its methods can be used to get or set any properties of the device that you may wish to examine or change before building the action map.

2. Obtaining the Default Action Map

[C++]
To obtain the default action map for the device, call **IDirectInputDevice8::BuildActionMap**. Microsoft® DirectInput® takes the list of virtual controls specified in your **DIACTIONFORMAT** structure and attempts to map these to physical device objects, returning the results in the same structure. You should examine the **dwHow** member of each **DIACTION** element to determine whether the control was successfully mapped. If it was, you can also ascertain what criterion was used in choosing the object—for example, configuration by the user or by the device manufacturer.

[Visual Basic]
To obtain the default action map for the device, call **DirectInputDevice8.BuildActionMap**. Microsoft® DirectInput® takes the list of virtual controls specified in your **DIACTIONFORMAT** type and attempts to map these to physical device objects, returning the results in the same type. You should examine the **lHow** member of each **DIACTION** element to determine whether the control was successfully mapped. If it was, you can also ascertain what criterion was used in choosing the object—for example, configuration by the user or by the device manufacturer.

3. Making Changes to the Action Map

[C++]
You now have the option of changing the default mappings, although it is not recommended that you do so. After examining the **dwSemantic** member of the **DIACTION** structure to determine which device object was mapped to an action, you can change that value. For example, if an action is mapped to DIJOFS_BUTTON9, but you want that action to be mapped to the trigger button instead, change the value to DIJOFS_BUTTON0 before applying the action map.

[Visual Basic]
You now have the option of changing the default mappings, although it is not recommended that you do so. After examining the **lSemantic** member of the **DIACTION** type to determine which device object was mapped to an action, you can change that value. For example, if an action is mapped to DIJOFS_BUTTON9, but you want that action to be mapped to the trigger button instead, change the value to DIJOFS_BUTTON0 before applying the action map.

4. Applying the Action Map

[C++]
When you are satisfied that the **DIACTIONFORMAT** structure contains suitable mappings for the device, call **IDirectInputDevice8::SetActionMap**. The value you assigned to the **uAppData** member of each **DIACTION** structure now becomes bound to the control specified in the **dwSemantic** member, which in turn is bound to a particular device object.

[Visual Basic]
When you are satisfied that the **DIACTIONFORMAT** type contains suitable mappings for the device, call **DirectInputDevice8.SetActionMap**. The value you assigned to the **lAppData** member of each **DIACTION** type now becomes bound to the control specified in the **lSemantic** member, which in turn is bound to a particular device object.

5. Mapping More than One Device

[C++]
Repeat steps 1 though 4 for each device you want to use in your application. Suppose you want to map actions to both a joystick and the keyboard. In the racing-game example, the action defined in the game as eB_DRIVERVIEW was mapped to a keyboard key in the following element of the DIACTION array.

```
{eB_DRIVERVIEW, DIKEYBOARD_1, "Driver View", },
```

In that example, when **BuildActionMap** is called on any device that is not a keyboard, the **lHow** member of the **DIACTION** structure for that element is set to DIAH_UNMAPPED. Continue examining the **lHow** member as each device in turn is enumerated, until a value other than DIAH_UNMAPPED is returned. This indicates that the device being currently mapped is a keyboard and the action has been successfully mapped to the requested key.

Even actions that have been successfully mapped can be mapped to another device. In the example, eB_UPSHIFT is given two **DIACTION** structures, as follows:

```
{eB_UPSHIFT, DIBUTTON_DRIVINGR_SHIFTUP, 0, "Upshift", },
...
{eB_UPSHIFT, DIKEYBOARD_PRIOR,        0, "Upshift", },
```

[Visual Basic]
Repeat steps 1 through 4 for each device you want to use in your application. Suppose you want to map actions to both a joystick and the keyboard. In the racing-game example, the action defined in the game as eB_DRIVERVIEW was mapped to a keyboard key in the DIACTION array.

In that example, when **BuildActionMap** is called on any device that is not a keyboard, the **lHow** member of the **DIACTION** type for that element is set to DIAH_UNMAPPED. Continue examining the **lHow** member as each device in turn is enumerated, until a value other than DIAH_UNMAPPED is returned. This indicates that the device being currently mapped is a keyboard and the action has been successfully mapped to the requested key.

Even actions that have been successfully mapped can be mapped to another device. In the example, eB_UPSHIFT is given two **DIACTION** types.

As devices are successively enumerated, the eB_UPSHIFT action is mapped to a suitable button on one or more joysticks or other game controllers, and then again to the keyboard.

6. Displaying the Configuration

[C++]
To show the user how actions have been mapped to devices, pass the DICD_DEFAULT flag to **IDirectInput8::ConfigureDevices**. The property sheet for the device, containing a graphical representation of mappings, is displayed in view-only mode as in the following diagram. For more information on the mechanics of displaying the image, refer to the Using Action Mapping tutorial.

[Visual Basic]
To show the user how actions have been mapped to devices, pass the DICD_DEFAULT flag to **DirectInput8.ConfigureDevices**. The property sheet for the device, containing a graphical representation of mappings, is displayed in view-only mode as in the following diagram. For more information on the mechanics of displaying the image, refer to the Using Action Mapping tutorial.

If the device manufacturer has not provided a device image, the mapping will be presented in text mode as in the following diagram.

Note

Even if the cooperative level for the application is disabling the Microsoft® Windows® logo key passively through an exclusive cooperative level or actively through use of the DISCL_NOWINKEY flag, that key will be active while the default action mapping UI is displayed.

For more information about this property sheet, see User Configuration of the Device.

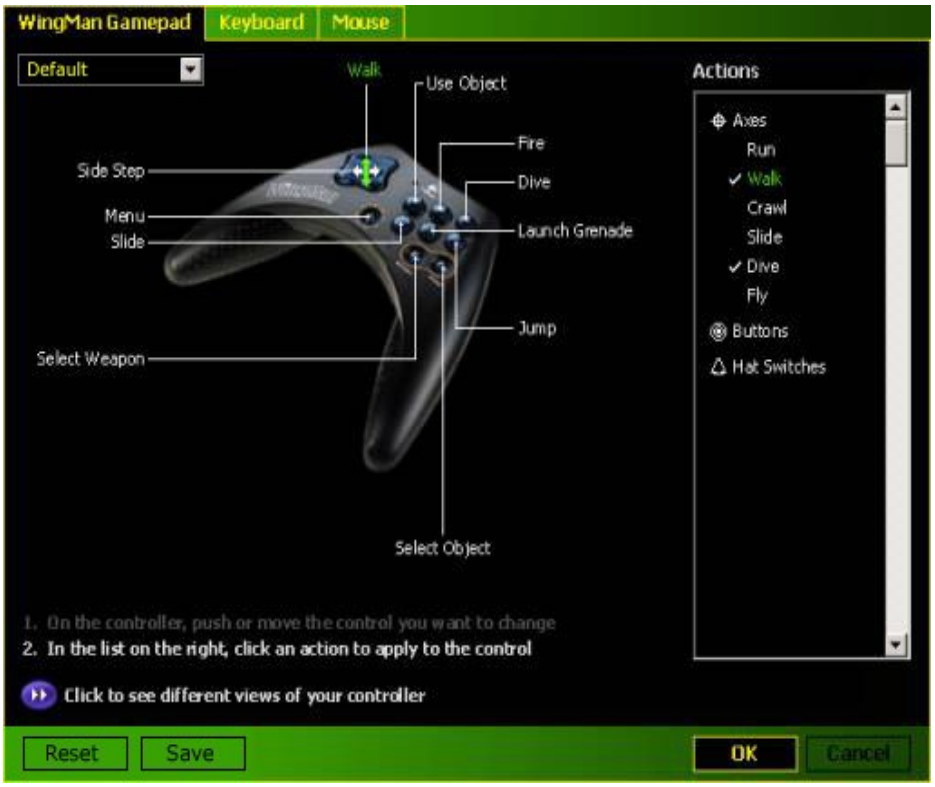
User Configuration of the Device

Microsoft® DirectInput® provides a property sheet that can be called from an application, enabling the user to configure devices for the application and view the current configuration. This property sheet can display various views of the device as provided by the manufacturer.

[C++]
To enable user configuration, pass a **DICONFIGUREDEVICESPARAMS** structure containing a pointer to the **DIACTIONFORMAT** structure describing the desired mapping, along with the DICD_EDIT flag, to the **IDirectInput8::ConfigureDevices** method. Normally you would do this after calling **IDirectInputDevice8::BuildActionMap** on all devices that will be used in the application.

[Visual Basic]
To enable user configuration, pass a **DICONFIGUREDEVICESPARAMS** type containing the **DIACTIONFORMAT** type describing the desired mapping, along with the DICD_EDIT flag, to the **DirectInput8.ConfigureDevices** method. Normally you would do this after calling **DirectInputDevice8.BuildActionMap** on all devices that will be used in the application.

The following illustration shows a typical property sheet in edit mode.



If the device manufacturer has not provided a device image, the mapping will be presented in text mode as in the following diagram.

Note

Even if the cooperative level for the application is disabling the Microsoft® Windows® logo key passively through an exclusive cooperative level or actively through use of the DISCL_NOWINKEY flag, that key will be active while the default action mapping UI is displayed.

[C++]

The property page for a device lists the friendly names that were provided by you in the **lptszActionName** member of each **DIACTION** structure. If you have already called **BuildActionMap** for a device, the page also shows these names as callouts on the image of the device, with lines pointing to the device objects to which the actions have been mapped.

The user now has the opportunity to reassign game actions by first choosing a control then choosing an action from the menu. When the user closes the property sheet, the method returns and the modifications are stored in the **DIACTIONFORMAT** structure that you passed in. You can now pass the same structure to

IDirectInputDevice8::SetActionMap in order to implement the new mapping scheme.

[Visual Basic]
The property page for a device lists the friendly names that were provided by you in the **ActionName** member of each **DIACTION** type. If you have already called **BuildActionMap** for a device, the page also shows these names as callouts on the image of the device, with lines pointing to the device objects to which the actions have been mapped.

The user now has the opportunity to reassign game actions by first choosing a control then choosing an action from the menu. When the user closes the property sheet, the method returns and the modifications are stored in the **DIACTIONFORMAT** type that you passed in. You can now pass the same type to **DirectInputDevice8.SetActionMap** in order to implement the new mapping scheme.

Retrieving Action Data

[C++]
You retrieve buffered data from action-mapped devices just as you would from unmapped devices: by calling **IDirectInputDevice8::GetDeviceData**. However, instead of identifying device objects by examining the **dwOfs** member of the **DIDEVICEOBJECTDATA** structure, you obtain the action associated with the object from the **uAppData** member. This is the same value you passed to the device in the **DIACTION** structure. It can be a simple identifier or a pointer to a function designed to handle the action.

[Visual Basic]
You retrieve buffered data from action-mapped devices just as you would from unmapped devices: by calling **DirectInputDevice8.GetDeviceData**. However, instead of identifying device objects by examining the **lOfs** member of the **DIDEVICEOBJECTDATA** type, you obtain the action associated with the object from the **lUserData** member. This is the same value you passed to the device in the **DIACTION** type.

Remember that an action can be associated with more than one device. You still have to obtain data from both devices independently, but you can use the same routine to handle the data regardless of where it comes from.

[C++]

The following sample code, which might be part of the game loop in a driving simulation, retrieves data from all devices in the *g_lpIdiDevices* array. This array contains *g_nDevices* elements.

```
for (int iDevice = 0x0; iDevice < g_nDevices; iDevice++)
{
   DIDEVICEOBJECTDATA didod;
   DWORD dwObjCount = 1;

   // Poll the device for data.
   g_lpDiDevices[iDevice]->Poll();

   // Retrieve the data.
   g_lpDiDevices[iDevice]->GetDeviceData( sizeof(didod),
                                          &didod,
                                          &dwObjCount, 0 );

   // Handle the actions regardless of what device returned them.
   switch(didod.uAppData)
   {
      case eA_STEER:
         SteerCar(didod.dwData);
         break;
      case eB_UPSHIFT
         if (didod.dwData & 0x80) ShiftGears(UPSHIFT);
         break;
      .
      .
      .

      default:
         break;
   }
}
```

Note

Axis constants for specific genres, such as DIAXIS_DRIVINGR_STEER or DIAXIS_SPACESIM_LATERAL, are used for absolute joystick data. The action mapper attempts to map this virtual control to a device object that returns absolute data. The data returned from that device should be processed accordingly in the application. Device constants such as DIMOUSE_XAXIS, however, are expected to return relative data.

When retrieving data, each potential source of data should be processed separately to keep one device object from possibly overwriting the data from another. For instance, the following **DIACTION** structures are used in an action map to control direction.

```
{INPUT_LEFTRIGHT_ABS_AXIS, DIAXIS_SPACESIM_LATERAL, 0,   _T("Turn"),},
```

```
{INPUT_LEFTRIGHT_REL_AXIS, DIMOUSE_XAXIS,          0,   _T("Turn"),},
{INPUT_TURNLEFT,          DIKEYBOARD_LEFT,     0,  _T("Turn left"),},
{INPUT_TURNRIGHT,         DIKEYBOARD_RIGHT,    0, _T("Turn right"),},
```

The application's input loop processes data from these actions in the following case statement.

```
switch (adod[j].uAppData)
{
   case INPUT_LEFTRIGHT_ABS_AXIS:
      g_dwAbsLR = adod[j].dwData
      break;
   case INPUT_LEFTRIGHT_REL_AXIS:
      g_dwRelLR = adod[j].dwData;
      break;
   case INPUT_TURNLEFT:
      g_bLeft = (adod[j].dwData != 0);
      break;
   case INPUT_TURNRIGHT:
      g_bRight = (adod[j].dwData != 0)
      break;
}
```

Note that each data source is assigned to a separate variable rather than all data sources being assigned a generic "turn" variable. If they were to share a generic variable, holding down the LEFT ARROW key and then moving the joystick would cause the keyboard information to be lost. This is because the joystick data would overwrite the variable.

In addition to individual variables, there are many ways to process the data. Whatever method is used, care should be taken in the processing of data to avoid unexpectedly lost information.

[Visual Basic]

The following sample code, which might be part of the game loop in a driving simulation, retrieves data from all devices in the *m_Devices* array. This array contains *m_nDevices* elements.

```
Dim iDevice As Integer
For iDevice = 1 To m_nDevices
   Dim didod(BUFFERSIZE) As DIDEVICEOBJECTDATA

   'Poll the device for data.
   m_Devices(iDevice).Poll

   'Determine how many pieces of data are available.
```

```
    m_nItems = m_Devices(iDevice).GetDeviceData(didod, 0)

    'Handle the actions regardless of what device returned them.
    For i = 0 To m_nItems
       Select Case didod(i).lUserData
          Case eA_STEER:
             Call SteerCar (didod(i).lData)
          Case eB_UPSHIFT:
             If (didod(i).lData & &H80) Then
                Call ShiftGears (UPSHIFT)
          .
          .
          .
       End Select
    Next
Next
```

Note

Axis constants for specific genres, such as DIAXIS_DRIVINGR_STEER or DIAXIS_SPACESIM_LATERAL, are used for absolute joystick data. The action mapper attempts to map this virtual control to a device object that returns absolute data. The data returned from that device should be processed accordingly in the application. Device constants such as DIMOUSE_XAXIS, however, are expected to return relative data.

When retrieving data, each potential source of data should be processed separately to keep one device object from possibly overwriting the data from another. For instance, the following **DIACTION** types are used in an action map to control direction.

```
With rgActions(0)
  .lAppData = INPUT_LEFTRIGHT_ABS_AXIS
  .lSemantic = DIAXIS_SPACESIM_LATERAL
  .lFlags = 0
  .ActionName = "Turn"
End With
With rgActions(1)
  .lAppData = INPUT_LEFTRIGHT_REL_AXIS
  .lSemantic = DIMOUSE_XAXIS
  .lFlags = 0
  .ActionName = "Turn"
End With
With rgActions(2)
  .lAppData = INPUT_TURNLEFT
  .lSemantic = DIKEYBOARD_LEFT
  .lFlags = 0
  .ActionName = "Turn left"
End With
With rgActions(3)
```

```
  .lAppData = INPUT_TURNRIGHT
  .lSemantic = DIKEYBOARD_RIGHT
  .lFlags = 0
  .ActionName = "Turn right"
End With
```

The application's input loop processes data from these actions in the following case statement.

```
Select Case adod(j).lUserData
   Case INPUT_LEFTRIGHT_ABS_AXIS:
      g_dwAbsLR = adod(j).lData
   Case INPUT_LEFTRIGHT_REL_AXIS:
      g_dwRelLR = adod(j).lData
   Case INPUT_TURNLEFT:
      g_bLeft = (adod(j).lData <> 0)
   case INPUT_TURNRIGHT:
      g_bRight = (adod(j).lData <> 0)
End Select
```

Note that each data source is assigned to a separate variable rather than all data sources being assigned a generic "turn" variable. If they to share a generic variable, holding down the LEFT ARROW key and then moving the joystick would cause the keyboard information to be lost. This is because the joystick data would overwrite the variable.

In addition to individual variables, there are many ways to process the data. Whatever method is used, care should be taken in the processing of data to avoid unexpectedly lost information.

Maintaining Files During Development

During a development cycle, unused and out of date .ini files may accumulate due to frequent action map changes, Microsoft® DirectX® reinstallations, multiple users, and other normal development situations. These files could possibly cause unexpected mappings or reports of "recent" (DIEDBS_RECENTDEVICE) for devices that would not be expected to return that value. For this reason, it is good practice to occasionally delete any unused .ini files. These files can be found in C:\Program Files\Common Files\DirectX\DirectInput\User Maps.

Note

The procedure suggested above is meant to be performed only manually during a development cycle to ensure that the development environment is in a cleaner state. A shipping application should never delete user maps as this could result in the loss of a user's preferred settings.

Force Feedback

Force feedback is the generation of push or resistance in an input/output device—for example, by motors mounted in the base of a joystick. Microsoft® DirectInput® enables you to generate force-feedback effects for devices that have compatible drivers.

The following sections introduce the elements of force feedback.

- Basic Concepts of Force Feedback
- Effect Enumeration
- Loading Effects from a File
- Information About a Supported Effect
- Creating an Effect
- Effect Direction
- Envelopes and Offsets
- Effect Playback
- Downloading and Unloading Effects
- Changing an Effect
- Gain
- Force-Feedback State
- Effect Object Enumeration
- Effect Types

Basic Concepts of Force Feedback

A particular instance of force feedback is called an *effect*, and the push or resistance is called the *force*. Most effects fall into one of the following categories:

- Constant force. A steady force in a single direction.
- Ramp force. A force that steadily increases or decreases in magnitude.
- Periodic effect. A force that pulsates according to a defined wave pattern.
- Condition. A reaction to motion or position along an axis. Two examples are a friction effect that generates resistance to movement of the joystick, and a spring effect that pushes the stick back toward a certain position after it has been moved from that position.

The strength of the force is called its *magnitude*. Magnitude is measured in units ranging from 0 (no force) through 10,000 (maximum force for the device, defined for C/C++ and Visual Basic as DI_FFNOMINALMAX). A negative value indicates

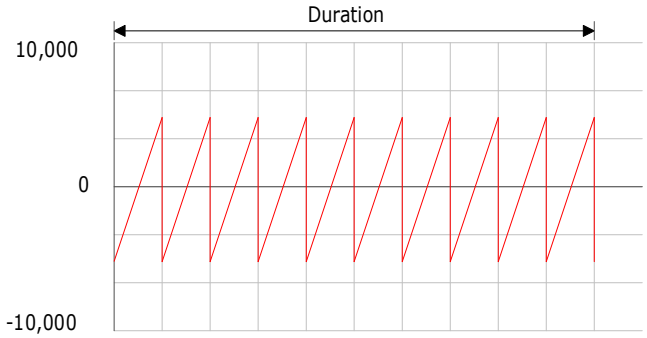
force in the opposite direction. Magnitudes are linear: a force of 10,000 is twice as great as one of 5,000.

Ramp forces have a beginning and ending magnitude. For a periodic effect, the basic magnitude is the force at the peak of the wave.

The *direction* of a force is the direction from which it comes. A positive force on a given axis pushes from the positive toward the negative.

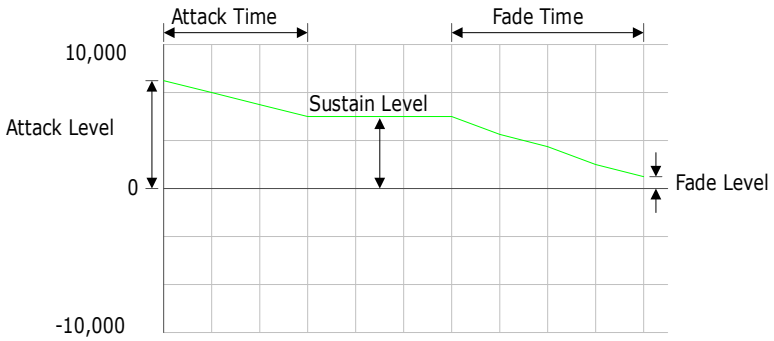
Effects also have *duration*, measured in microseconds. Periodic effects have a *period*, or the duration of one cycle, also measured in microseconds. The *phase* of a periodic effect is the point along the wave at which playback begins.

The following illustration represents a sawtooth periodic effect with a magnitude of 5,000, or half the maximum force for the device. The horizontal axis represents the duration of the effect, and the vertical axis represents the magnitude. Points above the center line represent positive force in the direction defined for the effect, and points below the center line represent negative force, or force in the opposite direction.

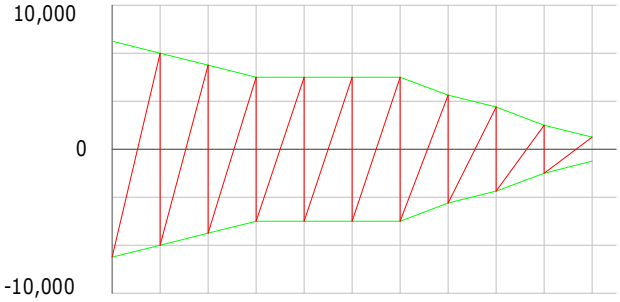


A force may be further shaped by an *envelope*. An envelope defines an *attack value* and a *fade value*, which modify the beginning and ending magnitude of the effect. Attack and fade also have duration, which determines how long the magnitude takes to reach or fall away from the *sustain value*, the magnitude in the middle portion of the effect.

The following diagram shows an envelope. The attack level is set to 8,000 and the fade level to 1,000. The sustain level is defined by the basic magnitude of the force to which the envelope is being applied; in the example, it is 5,000. In this case, the attack is greater than the sustain, giving the effect an initial strong kick. Both the attack and the fade level can be either greater or less than the sustain level.



The next diagram shows the result of the envelope being applied to the periodic effect in the first diagram. The envelope is mirrored on the negative side of the magnitude. An attack value of 8,000 means that the initial magnitude of the force in either direction is 80 percent of the maximum possible.



Periodic effects and conditions can also be modified by the addition of an *offset*, which defines the amount by which the waveform is shifted up or down from the base level. The practical effect of applying a positive offset to the sawtooth example would be to strengthen the positive force and weaken the negative one—in other words, the force would peak more strongly in one direction than in the other.

Finally, the overall magnitude of an effect can be scaled by *gain*, which is analogous to a volume control in audio. A single gain value can be applied to all effects for a device. You might want to do this to compensate for stronger or weaker forces on different hardware or to accommodate the user's preferences.

Effect Enumeration

[C++]
The **IDirectInputDevice8::EnumEffects** method returns information about the support offered by the device for various kinds of effects.

It is important to distinguish between *supported effects* and *created effects*, or *effect objects*. A supported effect is a basic type which a device is capable of playing, such as a constant force. However, a constant force by itself has no properties such as magnitude, direction, duration, attack, or fade. These properties are set when you *create* a specific effect object in your application by shaping the raw effect type with an envelope or setting the values of a DIEFFECT structure. A supported effect can be represented by many effect objects, each with different parameters—for example, several constant forces, each with different duration, magnitude, and direction.

For information on enumerating created effects, see Effect Object Enumeration.

Like other Microsoft® DirectInput® enumerations, the **IDirectInputDevice8::EnumEffects** method requires a callback function. This is documented with the placeholder name **DIEnumEffectsCallback**, but you can use a different name if you prefer. This function is called for each effect enumerated. Within the function you can obtain the GUID for each effect, get information about the extent of hardware support, and create one or more effect objects whose methods you can use to manipulate the effect.

The following code example sets the enumeration in motion. Assume that *g_lpdid* is an initialized pointer to the **IDirectInputDevice8** interface of the device.

```
HRESULT hr = g_lpdid->EnumEffects(&DIEnumEffectsCallback,
    g_lpdid, DIEFT_ALL);
```

A skeletal callback is shown in the following code. The *pvRef* parameter of the callback could be any 32-bit value, but in this case it is a pointer to the device interface (the global *g_lpdid* in the previous example), which is used for getting information about effects supported by the device and for creating effect objects.

```
BOOL CALLBACK DIEnumEffectsCallback(LPCDIEFFECTINFO pdei,
                    LPVOID pvRef)
{
  HRESULT           hr;
  LPDIRECTINPUTDEVICE8 lpdid = (LPDIRECTINPUTDEVICE8)pvRef;
                                                  // Pointer to calling device
  LPDIRECTINPUTEFFECT  lpdiEffect;    // Pointer to created effect
  DIEFFECT          diEffect;     // Params for created effect
  DICONSTANTFORCE     diConstantForce; // Type-specific parameters


  if (DIEFT_GETTYPE(pdei->dwEffType) == DIEFT_CONSTANTFORCE)
  {
    /* Here you can extract information about support for the
       effect type (from pdei), and tailor your effects
       accordingly. For example, the device might not support
       envelopes for this type of effect. */
    .
```

```
    .
    .
    // Create one or more constant force effects.
    // For each, you have to initialize a DICONSTANTFORCE
    // and a DIEFFECT structure.
    .
    .
    .
     hr = lpdid->CreateEffect(pdei->guid,
                              &diEffect,
                              &lpdiEffect,
                              NULL);
    .
    .
    .
  }
  // And so on for other types of effect
 .
 .
 .

  return DIENUM_CONTINUE;
} // End of callback
```

For a detailed example of creating an effect, and for more information on how to initialize an effect, see Creating an Effect.

[Visual Basic]

The **DirectInputDevice8.GetEffectsEnum** method enumerates effects supported by the device. It returns a **DirectInputEnumEffects** object representing the collection of supported effects. Methods of **DirectInputEnumEffects** can be used to get information about a particular effect.

It is important to distinguish between *supported effects* and *created effects*, or *effect objects*. A supported effect might be a constant force that can be shaped by an envelope. However, this effect has no properties such as magnitude, direction, duration, attack, or fade. You set these properties when you create an effect object in your application. A supported effect can be represented by many effect objects, each with different parameters—for example, several constant forces, each with different duration, magnitude, and direction.

The following code example shows how an application could enumerate hardware-specific effects, looking for a particular effect supported by the Microsoft® SideWinder® joystick. If the desired effect is not found, the application substitutes one of the standard effects:

```
Const BasketballDribble = _
```

```
      "{E84CD1AC-81FA-11D0-94AB-0080C74C7E95}"
Dim diEnumEffects As DirectInputEnumEffects
Dim EffGuid As String
Dim i As Integer

' diDev is a DirectInputDevice object.
Set diEnumEffects = didev.GetEffectsEnum(DIEFT_HARDWARE)
For i = 1 To diEnumEffects.GetCount
  If diEnumEffects.GetEffectGuid(i) = BasketballDribble Then
    EffGuid = BasketballDribble
    Exit For
  End If
Next i
If EffGuid <> BasketballDribble Then
  EffGuid = "GUID_Sine"
  ' Set parameters for emulated dribble here
  .
  .
  .
End If
```

Ultimately, *EffGuid* is passed to **DirectInputDevice8.CreateEffect**.

Loading Effects from a File

[C++]
Using the Force Editor supplied with the Microsoft® DirectX® SDK, or another application that uses the same file format, you can design effects and save them in a file. You can then use these effects in any application by loading them at run time.

To load effects, call the **IDirectInputDevice8::EnumEffectsInFile** method. By default, the enumeration is limited to the standard Microsoft DirectInput® effects, but you can enumerate other effects by setting the DIFEF_INCLUDENONSTANDARD flag. By setting the DIFEF_MODIFYIFNEEDED flag, you can also instruct DirectInput to modify the parameters of effects, if necessary, so that they work on the device. (For example, an effect authored for two axes can be made to work on a single-axis steering wheel.)

In the following code example, the first three standard effects are loaded from a file and created as DirectInputEffect objects.

```
// g_lpdid is a valid IDirectInputDevice8 pointer to a
// force feedback device.
```

```
// The array of effect pointers is declared globally.
LPDIRECTINPUTEFFECT pEff[3];
.
.
.
g_lpdid->EnumEffectsInFile("FEdit1.ffe",
                           EnumEffectsInFileProc,
                           NULL,
                           DIFEF_MODIFYIFNEEDED);
.
.
.
```

The following callback function is called once for each effect in the file or until it returns DIENUM_STOP. Note that because the DIFEF_MODIFYIFNEEDED flag was passed, the effect parameters may have been modified in *lpdife->lpDiEffect*. The callback creates the effect with these modified parameters.

```
BOOL CALLBACK EnumEffectsInFileProc(LPCDIFILEEFFECT lpdife,
    LPVOID pvRef)

{
   HRESULT     hr;
   static int  i;


   hr = g_lpdid->CreateEffect(lpdife->GuidEffect,
                              lpdife->lpDiEffect,
                              &pEff[i],
                              NULL);
   if (FAILED(hr))
   {
      // Error handling
   }
   if (++i > 2) return DIENUM_STOP;
   else return DIENUM_CONTINUE;
}
```

[Visual Basic}

Using the Force Editor supplied with the Microsoft® DirectX® SDK, or another application that uses the same file format, you can design effects and save them in a file. You can then use these effects in any application by loading them at run time.

To load effects, call the DirectInputDevice8.CreateEffectFromFile method. The resulting DirectInputEffect object may then be used normally.

Information About a Supported Effect

[C++]
The **IDirectInputDevice8::GetEffectInfo** method can be used to retrieve information about the device's support for an effect whose GUID is known. It retrieves the same information that is returned in the **DIEFFECTINFO** structure during enumeration. For more information, see Effect Enumeration.

The following code fetches information about an effect whose GUID is stored in the *EffectGuid* variable, and determines whether the direction of the effect can be changed without stopping and restarting it:

```
DIEFFECTINFO diEffectInfo;
diEffectInfo.dwSize = sizeof(DIEFFECTINFO);
g_lpdid->GetEffectInfo(&diEffectInfo, EffectGuid);
if (diEffectInfo.dwDynamicParams & DIEP_DIRECTION)
{
  // Can reset parameter dynamically
}
```

[Visual Basic]
In Microsoft® DirectX® for Microsoft Visual Basic®, information about a supported effect must be obtained from the **DirectInputEnumEffects** enumeration object. See Effect Enumeration.

The following code example gets information about the first enumerated effect and determines whether the direction of the effect can be changed without stopping and restarting it. Assume that *diEnumEffects* is an initialized **DirectInputEnumEffects** object.

```
Dim params As Long
params = diEnumEffects.GetDynamicParams(1)
If params And DIEP_DIRECTION Then
  Debug.Print "Direction is dynamic."
End If
```

Creating an Effect

[C++]
Create an effect object by using the **IDirectInputDevice8::CreateEffect** method, as in the following code example, where *g_lpdid* points to an instance of the interface. This example creates a very simple effect that pulls the joystick away from the user at full force for half a second.

```
HRESULT  hr;
LPDIRECTINPUTEFFECT lpdiEffect;  // receives pointer to created effect
DIEFFECT diEffect;               // parameters for created effect

DWORD    dwAxes[2] = { DIJOFS_X, DIJOFS_Y };
LONG     lDirection[2] = { 18000, 0 };

DICONSTANTFORCE diConstantForce;

diConstantForce.lMagnitude = DI_FFNOMINALMAX;   // Full force

diEffect.dwSize         = sizeof(DIEFFECT);
diEffect.dwFlags        = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
diEffect.dwDuration     = (DWORD)(0.5 * DI_SECONDS);
diEffect.dwSamplePeriod  = 0;               // = default
diEffect.dwGain         = DI_FFNOMINALMAX;   // No scaling
diEffect.dwTriggerButton = DIEB_NOTRIGGER;    // Not a button response
diEffect.dwTriggerRepeatInterval = 0;        // Not applicable
diEffect.cAxes                  = 2;
diEffect.rgdwAxes               = &dwAxes[0];
diEffect.rglDirection           = &lDirection[0];
diEffect.lpEnvelope             = NULL;
diEffect.cbTypeSpecificParams    = sizeof(DICONSTANTFORCE);
diEffect.lpvTypeSpecificParams   = &diConstantForce;

hr = g_lpdid->CreateEffect(GUID_ConstantForce,
                    &diEffect,
                    &lpdiEffect,
                    NULL);
```

In the method call, the first parameter identifies the supported effect with which the created effect is to be associated. The example uses one of the predefined GUIDs found in Dinput.h. If you use a predefined GUID, the call fails if the device does not support the effect.

The second parameter sets the parameters as specified in the **DIEFFECT** structure.

The third parameter receives a pointer to the effect object if the call is successful.

The DIEFF_POLAR flag specifies the type of coordinates used for the direction of the force. (See Effect Direction.) It is combined with DIEFF_OBJECTOFFSETS, which indicates that any buttons or axes used in other members are identified by their offsets within the **DIDATAFORMAT** structure for the device. The alternative is to use the DIEFF_OBJECTIDS flag, signifying that buttons and axes are identified by the **dwType** member of the **DIDEVICEOBJECTINSTANCE** structure returned for the object when it was enumerated with the **IDirectInputDevice8::EnumObjects** method.

For more information on the members of the **DIEFFECT** structure, see Effect Direction.

[Visual Basic]
Create an effect object by using the **DirectInputDevice8.CreateEffect** method, as in the following code example, where *didev* is a **DirectInputDevice8** object. This example creates a very simple effect that pulls the joystick away from the user at full force for half a second.

```
Dim effectInfo As DIEFFECT
Dim objDIEffect As DirectInputEffect

With effectInfo
  .constantForce.lMagnitude = 10000
  .lGain = 10000
  .lDuration = 500000
  .x = 18000
  .lTriggerButton = -1  ' No trigger button
End With

didev.Acquire
Set objDIEffect = didev.CreateEffect("GUID_ConstantForce", effectInfo)
```

The first parameter to **CreateEffect** can be one of the predefined GUID aliases, or an actual GUID in string form known from hardware documentation or retrieved for an enumerated effect by using the **DirectInputEnumEffects.GetEffectGuid** method.

The relevant members of the **DIEFFECT** type vary according to the kind of effect. Constant forces are the simplest kind, requiring only a single type-specific parameter. The **lGain**, **lDuration**, and **lTriggerButton** members should be set for all effects because the default values of 0 are not usually suitable.

By default, the direction of the effect is expressed in polar coordinates, meaning that **DIEFFECT.x** holds the direction from which the force comes, in hundredths of a degree, and **DIEFFECT.y** must be 0.

Effects are automatically downloaded to the device when created, provided the device is not full and is acquired at the exclusive cooperative level.

Effect Direction

[C++]
Directions can be defined for one or more axes. As with the mouse and joystick, the x-axis increases from left to right, and the y-axis increases from far to near. For three-dimensional devices, the z-axis increases from up to down.

The direction of an effect is the direction from which it comes. An effect with a direction along the negative y-axis tends to push the stick along the positive y-axis (toward the user). If the user must push the stick toward the left to counteract an effect, the effect has a left direction; that is, it lies on the negative x-axis.

Direction can be expressed in *polar, spherical,* or *Cartesian* coordinates.

Polar coordinates are expressed as a single angle, in hundredths of a degree clockwise from whatever zero-point, or true north, has been established for the effect. Normally this is the negative y-axis; that is, away from the user. Thus an effect with a polar coordinate of 9,000 has a direction of east, or to the user's right, and the user must exert force to the right to counteract it.

Spherical coordinates are also in hundredths of a degree but can contain two or more angles For each angle, the direction is rotated in the positive direction of the next axis. For a 3-D device, the first angle would normally be rotated from the positive x-axis toward the positive y-axis (clockwise from east), and the second angle would be rotated toward the positive z-axis (down). Thus a force with a direction of (0, 0) would be to the user's right and parallel to the tabletop. A direction of 27,000 for the first angle and 4,500 for the second would be directly away from the user (270 degrees clockwise from east) and angling toward the floor (45 degrees downward from the tabletop); to counteract a force with this direction, the user would have to push forward and down.

Cartesian coordinates are similar to 3-D vectors. If you draw a straight line on graph paper with an origin of (0, 0) at the center of the page, the direction of the line can be defined by the coordinates of any intersection that it crosses, regardless of the distance from the origin. A direction of (1, –2) and a direction of (5, –10) are exactly the same.

Note

The coordinates used in creating force-feedback effects define only direction, not magnitude or distance.

When an effect is created or modified, the **cAxes**, **rgdwAxes**, and **rglDirection** members of the **DIEFFECT** structure are used to specify the direction of the force.

The **cAxes** member specifies the number of axes involved in the effect. This will also be the number of elements in each of the arrays pointed to by the next two members.

The array pointed to by **rgdwAxes** identifies the axes to which the effects will be applied. If the DIEFF_OBJECTOFFSETS flag has been set, the axes are identified by the offsets within the data format structure. These offsets are most readily identified by using the DIJOFS_* defines. (For a list of these values, see Joystick Device Constants.)

Finally, the **rglDirection** member specifies the direction of the force.

Note

The **cAxes** and **rgdwAxes** members cannot be modified once they have been set. An effect always has the same axis list.

Regardless of whether you are using Cartesian, polar, or spherical coordinates, you must provide exactly as many elements in **rglDirection** as there are axes in the array pointed to by **rgdwAxes**.

In the polar coordinate system, north (0 degrees) lies along the vector (0, –1), where the elements of the vector correspond to the elements in the axis list pointed to by **rgdwAxes**. Normally those axes are x and y, so north is directly along the negative y-axis; that is, away from the user. The last element of *rglDirection* must be 0.

In the example in Creating an Effect, the direction of a two-dimensional force is defined in polar coordinates. The force has a south direction—it comes from the direction of the user, so the user has to pull the stick to counteract it. The direction is 180 degrees clockwise from north, and can be assigned as follows:

```
LONG  lDirection[2] = { 18000, 0 };
```

For greater clarity, the assignment could also be expressed this way:

```
LONG  lDirection[2] = { 180 * DI_DEGREES, 0 };
```

For spherical coordinates, presuming that you are working with a three-axis device, the same direction is assigned as follows:

```
LONG  lDirection[3] = { 90 * DI_DEGREES, 0, 0 }
```

The first angle is measured in hundredths of a degree from the (1, 0) direction, rotated in the direction of (0, 1); the second angle is measured in hundredths of a degree toward (0, 0, 1). The elements of the vector notation again correspond to elements in the array pointed to by the **rgdwAxes** member. Assume that the elements of this array represent the x, y, and z axes, in that order. The point of origin is at x = 1 and y = 0; that is, to the user's right. The direction of rotation is toward the positive y-axis (0, 1); that is, toward the user, or clockwise. The force in the example is 90 degrees clockwise from the right; that is, south. Because the second element of *lDirection* is 0, there is no rotation on the third axis.

How do you accomplish the same thing with Cartesian coordinates? Presuming that you have used the DIEFF_CARTESIAN flag in the **dwFlags** member, you would specify the direction like this:

```
LONG  lDirection[2] = { 0, 1 };
```

Here again, the elements of the array correspond to the axes listed in the array pointed to by **rgdwAxes**. The example sets the x-axis to 0 and the y-axis to 1; that is, the direction lies directly along the positive y-axis, or to the south.

The theory of effect directions can be difficult to grasp, but the practice is fairly straightforward. For code examples, see Examples of Setting Effect Direction.

[Visual Basic]
Microsoft® DirectX® for Microsoft Visual Basic® supports two-axis effects on the x-axis and y-axis.

The direction of an effect is the direction from which it comes. An effect with a direction along the negative y-axis tends to push the stick along the positive y-axis (toward the user). If the user must push the stick toward the left in order to counteract an effect, the effect has a left direction; that is, it lies on the negative x-axis.

Direction can be expressed in *polar* or *Cartesian* coordinates. By default, polar coordinates are used, and the direction is specified in the **x** member of the **DIEFFECT** type, with **y** always 0. To use Cartesian coordinates, you must specify DIEFF_CARTESIAN in the **lFlags** member and supply values in both **x** and **y**.

Polar coordinates are expressed as a single angle, in hundredths of a degree clockwise from whatever zero-point, or true north, has been established for the effect. Normally this is the negative y-axis; that is, away from the user. Thus an effect with a polar coordinate of 9,000 has a direction of east, or to the user's right, and the user must exert force to the right to counteract it.

Cartesian coordinates are similar to 3-D vectors and are most useful for matching a force to the user's orientation in a 3-D environment. If you draw a straight line on graph paper with an origin of (0, 0) at the center of the page, the direction of the line can be defined by the coordinates of any intersection that it crosses, regardless of the distance from the origin. A direction of (1, –2) and a direction of (5, –10) are exactly the same.

Note

The coordinates used in creating force-feedback effects define only direction, not magnitude or distance.

In the example in Creating an Effect, the direction of a two-dimensional force is defined in polar coordinates. The force has a south direction—it comes from the direction of the user, so the user has to pull the stick to counteract it. The direction is 180 degrees clockwise from north, and was assigned as follows, where *effectInfo* is a valid DIEFFECT type.

```
effectInfo.x = 18000
```

The **effectInfo.y** member must be set to 0, which it is by default. For greater clarity, the assignment could also be expressed this way:

```
effectInfo.x = 180 * DI_DEGREES
```

How do you accomplish the same thing with Cartesian coordinates? Presuming that you have used the DIEFF_CARTESIAN flag in the **lFlags** member, you would specify the direction like this:

```
effectInfo.x = 0
effectInfo.y = 1
```

As long as the **x** value is 0, any number in the **y** value will give the same effect because only direction is indicated, not magnitude. The theory of effect directions can be difficult to grasp, but the practice is fairly straightforward.

The following code example reverses the direction of a force represented by the **DirectInputEffect** object *dieff*, which was created using the global **DIEFFECT** type *effectInfo*:

```
effectInfo.lFlags = DIEFF_CARTESIAN
effectInfo.X = effectInfo.X * -1
effectInfo.Y = effectInfo.Y * -1
Call dieff.SetParameters(effectInfo, DIEP_DIRECTION)
```

For more code examples, see Examples of Setting Effect Direction.

Examples of Setting Effect Direction

[C++]

Single-Axis Effects

Setting up the direction for a single-axis effect is easy because there is nothing to specify. Put the DIEFF_CARTESIAN flag in the **dwFlags** member of the **DIEFFECT** structure and set **rglDirection** to point to a single **LONG** containing the value 0.

The following code example sets up the direction and axis parameters for an x-axis effect:

```
DIEFFECT eff;
LONG     lZero  = 0;                  // No direction
DWORD    dwAxis = DIJOFS_X;           // X-axis effect

ZeroMemory(&eff, sizeof(DIEFFECT));
eff.cAxes       = 1;                              // One axis
eff.dwFlags     = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS; // Flags
eff.rglDirection = &lZero;                        // Direction
eff.rgdwAxes     = &dwAxis;                       // Axis for effect
```

Two-Axis Effects with Polar Coordinates

Setting up the direction for a polar two-axis effect is only a little more complicated. Set the DIEFF_POLAR flag in **dwFlags** and set **rglDirection** to point to an array of two **LONG**s. The first element in this array is the direction from which you want the effect to come. The second element in the array must be 0.

The following example sets up the direction and axis parameters for a two-axis effect coming from the east:

```
DIEFFECT eff;
LONG     rglDirection[2] = { 90 * DI_DEGREES, 0 };
DWORD    rgdwAxes[2]     = { DIJOFS_X, DIJOFS_Y };   // X- and y-axis

ZeroMemory(&eff, sizeof(DIEFFECT));
eff.cAxes       = 2;                              // Two axes
eff.dwFlags     = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;  // Flags
eff.rglDirection = rglDirection;                    // Direction
eff.rgdwAxes     = rgdwAxes;                    // Axis for effect
```

Two-Axis Effects with Cartesian Coordinates

Setting up the direction for a Cartesian two-axis effect is a bit trickier. Set the DIEFF_CARTESIAN flag in **dwFlags**, and again set **rglDirection** to point to an array of two **LONG**s. This time the first element in the array is the x-coordinate of the direction vector, and the second is the y-coordinate.

The following code example sets up the direction and axis parameters for a two-axis effect coming from the east:

```
DIEFFECT eff;
LONG     rglDirection[2] = { 1, 0 };           // Positive x = east
DWORD    rgdwAxes[2]     = { DIJOFS_X, DIJOFS_Y };  // X- and y-axis

ZeroMemory(&eff, sizeof(DIEFFECT));
eff.cAxes = 2;                                 // Two axes
eff.dwFlags = DIEFF_CARTESIAN | DIEFF_OBJECTOFFSETS; // Flags
eff.rglDirection = rglDirection;                    // Direction
eff.rgdwAxes = rgdwAxes;                       // Axis for effect
```

[Visual Basic]

Single-Axis Effects with Cartesian Coordinates

Setting up the direction for a single-axis effect is easy because you need to specify only the direction of the axis. Put the DIEFF_CARTESIAN flag in the **lFlags** member of the DIEFFECT structure and set the **x** or **y** member to either 1 or -1, depending on the direction you want the effect to come from.

The following code example sets an x-axis effect with a right to left direction:

```
Dim eff As DIEFFECT

eff.lFlags = DIEFF_CARTESIAN Or DIEFF_OBJECTOFFSETS
eff.x = 1
eff.y = 0
```

Single-Axis Effects with Polar Coordinates

The same effect could be set up just as easily using polar coordinates. Put the DIEFF_POLAR flag in the **lFlags** member of the **DIEFFECT** structure and set the **x** member to either 0, 90, 180, or 270 degrees.

The following code example sets the same x-axis effect as in the previous example:

```
Dim eff As DIEFFECT

eff.lFlags = DIEFF_POLAR Or DIEFF_OBJECTOFFSETS
eff.x = 90 * DI_DEGREES
eff.y = 0
```

Two-Axis Effects with Polar Coordinates

Setting up the direction for a polar two-axis effect is no different than a polar one-axis effect. The only difference is that the range of degree values is not restricted to the four axis values.

The following code example sets an effect originating from the user's upper right:

```
Dim eff As DIEFFECT

eff.lFlags = DIEFF_POLAR Or DIEFF_OBJECTOFFSETS
eff.x = 45 * DI_DEGREES
eff.y = 0
```

Two-Axis Effects with Cartesian Coordinates

Setting up the direction for a Cartesian two-axis effect is straightforward as well. First, set the DIEFF_CARTESIAN flag in **lFlags**. The **x** and **y** members will be filled with the x and y coordinates of any point on a line of direction through that point and the origin (0,0).

The following code example sets the same effect as in the previous sample:

```
Dim eff As DIEFFECT

eff.lFlags = DIEFF_CARTESIAN Or DIEFF_OBJECTOFFSETS
eff.x = 1
eff.y = 2
```

Envelopes and Offsets

You can modify the basic magnitude of some effects by applying an envelope and an offset. For an overview, see Basic Concepts of Force Feedback.

[C++]
To apply an envelope when creating or modifying an effect, initialize a **DIENVELOPE** structure and put a pointer to it in the **lpEnvelope** member of the **DIEFFECT** structure.

The device driver determines which effects support envelopes. Typically, you can apply an envelope to a constant force, a ramp force, or a periodic effect, but not to a condition. To determine whether a particular effect supports an envelope, call the **IDirectInputDevice8::GetEffectInfo** method and check for the DIEP_ENVELOPE flag in the **dwStaticParams** member of the **DIEFFECTINFO** structure.

To apply an offset, set the **lOffset** member of the **DIPERIODIC** or **DICONDITION** structure pointed to by the **lpvTypeSpecificParams** member of the **DIEFFECT** structure. For periodic effects, the absolute value of the offset plus the magnitude of the effect must not exceed DI_FFNOMINALMAX.

[Visual Basic]
To apply an envelope when creating or modifying an effect, set the **bUseEnvelope** member of the **DIEFFECT** type to True and initialize the **envelope** member, which is a **DIENVELOPE** type.

The device driver determines which effects support envelopes. Typically, you can apply an envelope to a constant force, a ramp force, or a periodic effect, but not to a condition. To determine whether a particular effect supports an envelope, call the **DirectInputEnumEffects.GetStaticParams** method and check for the DIEP_ENVELOPE flag in the returned value.

To apply an offset, set the **lOffset** member of the **DIPERIODICFORCE** or **DICONDITION** type used in the **periodicForce**, **conditionX** or **conditionY** member of **DIEFFECT**. For periodic effects, the absolute value of the offset plus the magnitude of the effect must not exceed 10,000.

You cannot apply an offset to a constant force or ramp force. In these cases, you can achieve the same effect by altering the magnitude.

Effect Playback

[C++]
There are two principal ways to start playback of an effect: manually by a call to the **IDirectInputEffect::Start** method, or automatically in response to a button press. Playback also starts when you change an effect by calling the **IDirectInputEffect::SetParameters** method with the DIEP_START flag.

Passing INFINITE in the *dwIterations* parameter of the **IDirectInputEffect::Start** method has the effect of playing the effect repeatedly, with the envelope being applied each time. If you want to repeat an effect without repeating the envelope—for example, to begin with a strong kick, then settle down to a steady throb—set *dwIterations* to 1, and set the **dwDuration** member of the **DIEFFECT** structure to INFINITE. (This is the structure passed to the **IDirectInputDevice8::CreateEffect** method.)

Note
Some devices do not support multiple iterations of an effect, and they accept only the value 1 in the *dwIterations* parameter to the **Start** method. Always check the return value from **Start** to be sure the effect played successfully.

To associate an effect with a button press, set the **dwTriggerButton** member of the **DIEFFECT** structure. Also set the **dwTriggerRepeatInterval** member to the desired delay between playbacks when the button is held down. This is the interval, in microseconds, between the end of one playback and the start of the next.

Note
On some devices, multiple effects cannot be triggered by the same button. If you associate more than one effect with a button, the last effect downloaded is the one triggered. Also, trigger repeat interval might not be supported.

To dissociate an effect from its trigger button, either call the **IDirectInputEffect::Unload** method or set the parameters for the effect with **dwTriggerButton** set to DIEB_NOTRIGGER.

Triggered effects, like all others, are lost when the application loses access to the device. To make them active again, download them as soon as the application reacquires the device. See **Downloading and Unloading Effects**. This step is not necessary for effects not associated with a trigger, because they are automatically downloaded if necessary whenever the **Start** method is called.

If an effect has a finite duration and is started by a call to the **Start** method, it stops playing when the time has elapsed. If its duration was set to INFINITE, playback ends only when the **IDirectInputEffect::Stop** method is called. An effect associated with a trigger button starts when the button is pressed and stops when the button is released or the duration has elapsed, whichever comes sooner.

[Visual Basic]

There are two principal ways to start playback of an effect: manually by a call to the **DirectInputEffect.Start** method, or automatically in response to a button press. Playback also starts when you change an effect by calling the **DirectInputEffect.SetParameters** method with the DIEP_START flag.

Passing –1 in the *iterations* parameter of the **Start** method has the effect of playing the effect repeatedly, with the envelope being applied each time. If you want to repeat an effect without repeating the envelope—for example, to begin with a strong kick, then settle down to a steady throb—set *iterations* to 1 and set the **lDuration** member of the **DIEFFECT** type to –1. (This is the type passed to the **DirectInputDevice8.CreateEffect** method.)

Note

Some devices do not support multiple iterations of an effect and accept only the value 1 in the *iterations* parameter to the **Start** method.

To associate an effect with a button press, set the **lTriggerButton** member of the **DIEFFECT** type. Also set the **lTriggerRepeatInterval** member to the desired delay between playbacks when the button is held down. This is the interval, in microseconds, between the end of one playback and the start of the next.

Note

On some devices, multiple effects cannot be triggered by the same button; if you associate more than one effect with a button; the last effect downloaded is the one triggered. Also, trigger repeat interval might not be supported.

To dissociate an effect from its trigger button, either call the **DirectInputEffect.Unload** method or set the parameters for the effect with **lTriggerButton** set to -1 or DIEB_NOTRIGGER.

Triggered effects, like all others, are lost when the application loses access to the device. To make them active again, download them as soon as the application reacquires the device. This step is not necessary for effects not associated with a trigger because they are automatically downloaded if necessary whenever the **Start** method is called.

If an effect has a finite duration and is started by a call to the **Start** method, it stops playing when the time has elapsed. If an effect has been instructed to play continually, playback ends only when the **DirectInputEffect.Stop** method is called. An effect associated with a trigger button starts when the button is pressed and stops when the button is released or the duration has elapsed, whichever comes sooner.

Downloading and Unloading Effects

Before an effect can be played, it must be downloaded to the device. Downloading an effect means telling the driver to prepare the effect for playback. It is entirely up to the driver to determine how this is done. Generally, the driver places the parameters

of the effect in hardware memory to minimize the subsequent transfer of data between the device and the system. The consequent reduction in latency is particularly important for conditions and for effects played in response to a trigger, such as a fire button. Ideally the device does not have to communicate with the system at all in order to respond to axis movements and button presses.

Downloading is done automatically when you create an effect, provided the device is not full and is acquired at the exclusive cooperative level. By default, it is also done when you start the effect or change its parameters.

[C++]
If you specify the DIEP_NODOWNLOAD flag when changing parameters, you must subsequently use the **IDirectInputEffect::Download** method to download or update the effect.

When the device is unacquired—for example, when it has been acquired with the exclusive foreground cooperative level and the application moves to the background—effects are unloaded and must be downloaded again when the application regains the foreground. This is done automatically when you call the **IDirectInputEffect::Start** method, but you can choose to download all effects immediately on reacquiring the device. You always have to download effects associated with a trigger button because the **Start** method is not normally called for such effects.

If your application gets the DIERR_DEVICEFULL error when downloading an effect, you must make room for the new effect by unloading an old one. You can remove an effect from the device by calling the **IDirectInputEffect::Unload** method. You can also remove all effects by resetting the device through a call to the **IDirectInputDevice8::SendForceFeedbackCommand** method.

[Visual Basic]
If you specify the DIEP_NODOWNLOAD flag when changing parameters, you must subsequently use the **DirectInputEffect.Download** method to download or update the effect.

When the device is unacquired—for example, when it has been acquired with the exclusive foreground cooperative level and the application moves to the background—effects are unloaded and must be downloaded again when the application regains the foreground. This is done automatically when you call the **DirectInputEffect.Start** method, but you can choose to download all effects immediately on reacquiring the device. You always have to download effects associated with a trigger button because the **Start** method is not normally called for such effects.

If your application gets the DIERR_DEVICEFULL error when downloading an effect, you must make room for the new effect by unloading an old one. You can remove an effect from the device by calling the **DirectInputEffect.Unload** method. You can also remove all effects by resetting the device through a call to the **DirectInputDevice8.SendForceFeedbackCommand** method.

When you create a force-feedback device, the hardware and driver are reset, so any existing effects are cleared.

Changing an Effect

[C++]
You can modify the parameters of an effect, in some cases even while the effect is playing. Do this by using the **IDirectInputEffect::SetParameters** method.

The **dwDynamicParams** member of the **DIEFFECTINFO** structure tells you which effect parameters can be changed while an effect is playing. If you attempt to modify an effect parameter that cannot be modified while the effect is playing, and the effect is still playing, Microsoft® DirectInput® normally stops the effect, updates the parameters, and restarts the effect. You can override this default behavior by passing the DIEP_NORESTART flag.

The following code example changes the magnitude of the constant force that was set in the example under Creating an Effect.

```
DIEFFECT       diEffect;          // Parameters for effect
DICONSTANTFORCE diConstantForce;    // Type-specific parameters

diConstantForce.lMagnitude = 5000;
diEffect.dwSize = sizeof(DIEFFECT);
diEffect.cbTypeSpecificParams = sizeof(DICONSTANTFORCE);
diEffect.lpvTypeSpecificParams = &diConstantForce;
hr = lpdiEffect->SetParameters(&diEffect, DIEP_TYPESPECIFICPARAMS);
```

The DIEP_TYPESPECIFICPARAMS flag ensures that the transfer of data from the **DIEFFECT** structure is restricted to the relevant members, so that you do not have to initialize the entire structure and so that the minimum possible amount of data needs to be sent to the device.

[Visual Basic]
You can modify the parameters of an effect, in some cases even while the effect is playing. Do this by using the **DirectInputEffect.SetParameters** method.

To find which effect parameters can be changed while an effect is playing, call the **DirectInputEnumEffects.GetDynamicParams** method.

If you attempt to modify an effect parameter that cannot be modified while the effect is playing, and the effect is still playing, Microsoft® DirectInput® normally stops the effect, updates the parameters, and restarts the effect. You can override this default behavior by passing the DIEP_NORESTART flag to **SetParameters**.

The following code example changes the magnitude of a constant force represented by the **DirectInputEffect** object *dieff*, which was created using the global **DIEFFECT** type *effectinfo*:

```
effectinfo.constantForce.lMagnitude = 5000
Call dieff.SetParameters(effectinfo, DIEP_TYPESPECIFICPARAMS)
```

Note

You must set the DIEP_TYPESPECIFICPARAMS flag if you are changing the **condition**, **constantforce**, **periodicforce**, or **rampforce** members of **DIEFFECT**.

Gain

You might want to scale the force of your effects according to the force exerted by different devices. For example, if an application's effects feel right on a device that puts out a maximum force of *n* newtons on a given axis, you might want to adjust the gain for a device that puts out more force. (You cannot use the gain to increase the maximum force of the axis, so you should set the basic effect magnitudes to values suitable for devices that put out less force.)

Gain can also be used to decrease the magnitude of a hardware-defined effect.

[C++]
The actual force generated by a device object such as an axis or button is returned in the **dwFFMaxForce** member of the **DIDEVICEOBJECTINSTANCE** structure when objects are enumerated. (See Device Object Enumeration.)

You can set the gain for the entire device by using the **IDirectInputDevice8::SetProperty** method.

You must set the gain for individual effects when creating them by putting a value in the **dwGain** member of the **DIEFFECT** structure. (If **dwGain** is 0, the effect will not be felt.) You can change this value later by using **IDirectInputEffect::SetParameters**, passing DIEP_GAIN in the *dwFlags* parameter.

[Visual Basic]
You can set the gain for the entire device by using the **DirectInputDevice8.SetProperty** method.

You must set the gain for individual effects when creating them by putting a value in the **lGain** member of the **DIEFFECT** type. (If **lGain** is 0, the effect will not be felt.) You can change this value later by using **DirectInputEffect.SetParameters**, passing DIEP_GAIN in the *flags* parameter.

The purpose of setting the device gain is to allow your application to have control over the strength of all effects all at once. For example, you might have a slider control in your application to enable the user to specify how strong the force-feedback effects should be, such as the master volume control on a sound mixer. When the device gain is set, your application does not need to adjust the gain of each individual effect to suit the user's preferences.

A gain value can be in the range from 0 through 10,000, where 10,000 indicates that magnitudes are not to be scaled, 7,500 means that forces are to be scaled to 75 percent of their nominal magnitudes, and so on.

Force-Feedback State

[C++]
The **IDirectInputDevice8::SendForceFeedbackCommand** method enables you to turn off the device's actuators (effectively causing it to ignore any effects that are being played), pause, stop, or continue playback of effects, and reset the device so that all downloaded effects are removed.

To retrieve the current force-feedback state, use the **IDirectInputDevice8::GetForceFeedbackState** method. This method returns information about whether the actuators are active, whether playback is paused, and whether the device has been reset. It also retrieves information about various switches and whether the device is currently powered.

[Visual Basic]
The **DirectInputDevice8.SendForceFeedbackCommand** method enables you to turn off the device's actuators (effectively causing it to ignore any effects that are being played), pause, stop, or continue playback of effects, and reset the device so that all downloaded effects are removed.

To retrieve the current force-feedback state, use the **DirectInputDevice8.GetForceFeedbackState** method. This method returns information about whether the actuators are active, whether playback is paused, and whether the device has been reset. It also retrieves information about various switches and about whether the device is currently powered.

Effect Object Enumeration

[Visual Basic]
Enumeration of created effects is not supported under Microsoft® Visual Basic®.

[C++]

When you need to examine or manipulate all the effects you have created, you can use the **IDirectInputDevice8::EnumCreatedEffectObjects** method. As no flags are currently defined for this method, you cannot restrict the enumeration to particular kinds of effects; all effects will be enumerated.

Note

This method enumerates created effects, not effects supported by a device. For more information on the distinction between the two, see Effect Enumeration.

Like other Microsoft® DirectInput® enumerations, the **EnumCreatedEffectObjects** method requires a callback function. This standard callback is documented with the placeholder name **DIEnumCreatedEffectObjectsCallback**, but you can use a different name. The function is called for each effect enumerated. Within the function, you can perform any processing you want. However, it is not safe to create a new effect while enumeration is going on.

The following is a skeletal example of the callback function. The *pvRef* parameter of the callback can be any 32-bit value; in this case, it is a pointer to the **IDirectInputDevice8** interface that is performing the enumeration.

```
HRESULT  hr;

BOOL CALLBACK DIEnumCreatedEffectObjectsCallback(
                    LPDIRECTINPUTEFFECT peff, LPVOID pvRef);
{
   LPDIRECTINPUTDEVICE pdid = (IDirectInputDevice*)pvRef;
                        // Pointer to calling device
   DIEFFECT          diEffect;     // Params for created effect

   diEffect.dwSize = sizeof(DIEFFECTINFO);
   peff->GetParameters(&diEffect, DIEP_ALLPARAMS);
   // Check or set parameters, or do anything else.
   .
   .
   .
} // End of callback
```

Here's the call that sets the enumeration in motion, passing in the DirectInputDevice pointer *lpdid*:

```
hr = lpdid->EnumCreatedEffectObjects(
       &DIEnumCreatedEffectObjectsCallback,
       &lpdid, 0);
```

Effect Types

This section contains information specific to the following types of force-feedback effects:

- Constant Forces
- Ramp Forces
- Periodic Effects
- Conditions
- Custom Forces
- Device-Specific Effects

Constant Forces

A constant force is a force with a defined magnitude and duration.

You can apply an envelope to a constant force to give it shape. For example, suppose you have an effect with a nominal magnitude of 2,000 and a duration of 2 seconds. Then you apply an envelope with the following values:

Parameter	**Value**
Attack time	0.5 second
Initial attack level	5,000
Fade time	1 second
Fade level	0

When you play the effect, you get the following:

Elapsed time	**Magnitude**
0.0	5,000
0.1	4,400
0.2	3,800
0.3	3,200
0.4	2,600
0.5	2,000
(Duration of sustain)	2,000
1.0:	2,000
1.1	1,800
1.2	1,600
1.3	1,400
1.4	1,200
1.5	1,000

1.6	800
1.7	600
1.8	400
1.9	200
2.0	0

You cannot apply an offset to a constant force.

[C++]
To create a constant force, pass GUID_ConstantForce to the **IDirectInputDevice8::CreateEffect** method. You can also pass any other GUID obtained by the **IDirectInputDevice8::EnumEffects** method, provided the low byte of the **dwEffType** member of the **DIEFFECTINFO** structure (**DIEFT_GETTYPE**(**dwEfftype**)) is equal to DIEFT_CONSTANTFORCE. In this way, you can use hardware-specific forces designed by the manufacturer such as a "constant" force that actually varies in magnitude in a seemingly random fashion to simulate turbulence.

A constant force uses a **DICONSTANTFORCE** structure to define the magnitude. A negative magnitude has the effect of reversing the direction of the force.

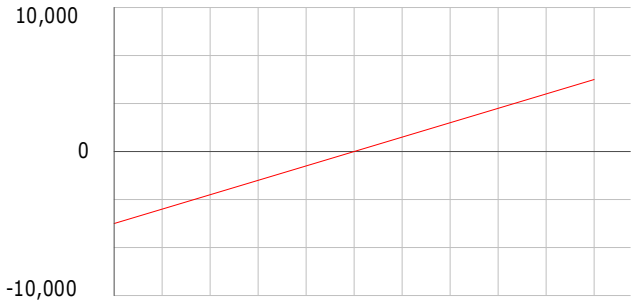
[Visual Basic]
To create a constant force, pass "GUID_ConstantForce" to the **DirectInputDevice8.CreateEffect** method. You can also pass any other GUID obtained by the **DirectInputEnumEffects.GetEffectGuid** method, provided the low byte of the value returned by **DirectInputEnumEffects.GetType** is equal to DIEFT_CONSTANTFORCE. In this way, you can use hardware-specific forces designed by the manufacturer such as a "constant" force that actually varies in magnitude in a seemingly random fashion to simulate turbulence.

The magnitude of a constant force is contained in a **DICONSTANTFORCE** type in the **constantForce** member of **DIEFFECT**. A negative magnitude has the effect of reversing the direction of the force.

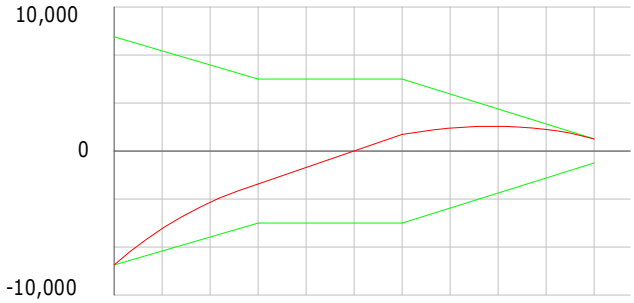
Ramp Forces

A ramp force is a force with defined starting and ending magnitudes and a finite duration. A ramp force can continue in a single direction, or it can start as a strong push in one direction, weaken, stop, and then strengthen in the opposite direction.

The following illustration shows a ramp force that starts at a magnitude of -5,000 and ends at 5,000:



You can apply an envelope to a ramp force to shape it further. The following diagram shows the effect of applying an envelope, shown in green, to the ramp force in the previous diagram.



Note that during the sustain portion of the envelope, the magnitude of the effect follows the same straight line as before the envelope was applied. For the duration of the attack and fade, the slope of the ramp is modified by the attack and fade levels.

You cannot apply an offset to a ramp force.

[C++]
To create a ramp force, pass GUID_RampForce to the **IDirectInputDevice8::CreateEffect** method. You can also pass any other GUID obtained by the **IDirectInputDevice8::EnumEffects** method, provided the low byte of the **dwEffType** member of the **DIEFFECTINFO** structure (**DIEFT_GETTYPE**(**dwEfftype**)) is equal to DIEFT_RAMPFORCE. In this way, you can use hardware-specific ramp forces designed by the manufacturer.

A ramp force uses a **DIRAMPFORCE** structure to define the starting and ending magnitude of the force, whereas the duration is taken from the **DIEFFECT** structure. Duration must never be set to INFINITE.

[Visual Basic]
To create a ramp force, pass "GUID_RampForce" to the **DirectInputDevice8.CreateEffect** method. You can also pass any other GUID obtained by the **DirectInputEnumEffects.GetEffectGuid** method, provided the low byte of the value returned by **DirectInputEnumEffects.GetType** is equal to DIEFT_RAMPFORCE. In this way, you can use hardware-specific ramp forces designed by the manufacturer.

The starting and ending magnitude of the force are defined in a **DIRAMPFORCE** type in the **rampForce** member of **DIEFFECT**. The duration cannot be infinite, and therefore **DIEFFECT.lDuration** must be a positive value.

Periodic Effects

Periodic effects are waveform effects. Microsoft® DirectInput® defines the following waveforms:

- Square.
- Sine.
- Cosine.
- Triangle.
- SawtoothUp. The waveform drops vertically after it reaches maximum positive force. See Basic Concepts of Force Feedback for an example.
- SawtoothDown. The waveform rises vertically after it reaches maximum negative force.

An envelope can be applied to periodic effects. See the example in Basic Concepts of Force Feedback.

The phase of a periodic effect is the point along the waveform where the effect begins. Phase is measured in hundredths of a degree, from 0 to 35,999. The following table indicates where selected phase values (in degrees) lie along the various waveforms. *Max* is the top (+) or bottom (–) of the wave, and *Mid* is the midpoint, where no force is applied in either direction.

Waveform	**0**	**90**	**180**	**270**
Square	+Max	+Max	–Max	–Max
Sine	Mid	+Max	Mid	–Max
Triangle	+Max	Mid	–Max	Mid
SawtoothUp	–Max	–Max/2	Mid	+Max/2 (reaches +Max just before the cycle repeats)
SawtoothDown	+Max	+Max/2	Mid	–Max/2 (reaches –Max just before the

cycle repeats)

A driver may round off a phase value to the nearest supported value. For example, for a sine effect some drivers support only values of 0 and 9,000 (to create a cosine); for other effects, only values of 0 and 18,000 are supported.

[C++]
To create a periodic force, pass one of the following values in the *rguid* parameter of the **IDirectInputDevice8::CreateEffect** method:

- GUID_Square
- GUID_Sine
- GUID_Triangle
- GUID_SawtoothUp
- GUID_SawtoothDown

You can also pass any other GUID obtained by the **IDirectInputDevice8::EnumEffects** method, provided the low byte of the **dwEffType** member of the **DIEFFECTINFO** structure (**DIEFT_GETTYPE**(**dwEfftype**)) is equal to DIEFT_PERIODIC. In this way, you can use hardware-specific forces designed by the manufacturer. For example, a hardware device might support a periodic effect that rotates the stick in a small circle.

The type-specific structure for periodic effects is **DIPERIODIC**.

Do not confuse the *period* of a periodic effect (**DIPERIODIC.dwPeriod**) with the *sample period (***DIEFFECT.dwSamplePeriod***)*. The period is the length of time that it takes to go through a complete wave cycle. The sample period, as for all effects, is the minimum time between actual adjustments of magnitude.

[Visual Basic]
To create a periodic force, pass one of the following values in the *guid* parameter of the **DirectInputDevice8.CreateEffect** method:

- "GUID_Square"
- "GUID_Sine"
- "GUID_Triangle"
- "GUID_SawtoothUp"
- "GUID_SawtoothDown"

You can also pass any other GUID obtained by the **DirectInputEnumEffects.GetEffectGuid** method, provided the low byte of the value returned by **DirectInputEnumEffects.GetType** is equal to DIEFT_PERIODIC. In this way, you can use hardware-specific periodic forces designed by the manufacturer.

The type-specific parameters for periodic effects are contained in a **DIPERIODICFORCE** type in the **periodicForce** member of **DIEFFECT**.

Do not confuse the period of a periodic effect (**DIEFFECT.periodic.lPeriod**) with the sample period (**DIEFFECT.lSamplePeriod**). The period is the length of time it takes to go through a complete wave cycle. The sample period, as for all effects, is the minimum time between actual adjustments of magnitude.

Conditions

Conditions are forces applied in response to current sensor values within the device. In other words, conditions require information about device motion, such as position or velocity of a joystick handle.

In general, conditions are not associated with individual events during a game or other application. They represent ambient phenomena, such as the stiffness or looseness of a flight stick, or the tendency of a steering wheel to return to a straight-ahead position.

A condition does not have a predefined magnitude. The magnitude is scaled in proportion to the movement or position of the input object.

Microsoft® DirectInput® defines the following types of condition effects:

- Friction. The force is applied when the axis is moved and depends on the defined friction coefficient.
- Damper. The force increases in proportion to the velocity with which the user moves the axis.
- Inertia. The force increases in proportion to the acceleration of the axis.
- Spring. The force increases in proportion to the distance of the axis from a defined neutral point.

Most hardware devices do not support the application of envelopes to conditions.

Conditions have the following type-specific parameters:

Offset
: The offset from the zero reading of the appropriate sensor value at which the condition begins to be applied. For a spring effect, the neutral point—that is, the point along the axis at which the spring would be considered at rest—would be defined by the offset for the condition. For a damper, the offset would define the greatest velocity value for which damping force is zero. Offset is not normally used for inertia or friction effects.

Coefficient
: A multiplier that scales the effect. For some devices, you can set separate coefficients for the positive and negative direction along the axis associated with the condition. For example, a flight stick controlling a damaged aircraft might move more easily to the right than to the left.

Saturation

In force feedback, saturation is an expression of the maximum possible force for an effect. For example, suppose a flight stick has a spring condition on the x-axis. The offset is 0 and the coefficient is 10,000, so the maximum force is normally exerted when the stick is furthest from the center. But if you define a positive and negative saturation of 5,000, the force does not increase after the stick has been moved halfway to the right or left.

Not all devices support saturation.

Deadband

The deadband is a zone around the offset of an axis at which the condition is not active. In the case of a spring that is at rest in the middle of an axis, the deadband enlarges this area of rest.

Not all devices support deadband.

Conditions can have duration, although in most cases you would probably want to set the duration to infinite (–1) and stop the effect only in response to some event in the application.

[C++]
To create a condition, pass one of the following values in the *rguid* parameter of the **IDirectInputDevice8::CreateEffect** method:

- GUID_Spring
- GUID_Damper
- GUID_Inertia
- GUID_Friction

You can also pass any other GUID obtained by the **IDirectInputDevice8::EnumEffects** method, provided the low byte of the **dwEffType** member of the **DIEFFECTINFO** structure (**DIEFT_GETTYPE**(**dwEfftype**)) is equal to DIEFT_CONDITION. In this way, you can use hardware-specific conditions designed by the manufacturer.

The type-specific structure for conditions is **DICONDITION**. For multiple-axis conditions, you can provide an array of such structures, one for each axis, or a single structure that defines the condition in the specified direction. In either case, you must set the **cbTypeSpecificParams** member of the **DIEFFECT** structure to the number of bytes used; that is, to **sizeof(DICONDITION)** * *n*, where *n* is the number of structures provided. For more information on how to use either single or multiple structures, see the **DICONDITION** structure.

An application should call the **IDirectInputDevice8::GetEffectInfo** method or the **IDirectInputDevice8::EnumEffects** method and examine the **dwEffectType** member of the **DIEFFECTINFO** structure to determine whether both a positive and a negative coefficient and saturation for the effect are supported on the device. If the effect does not return the DIEFT_POSNEGCOEFFICIENTS flag, it ignores the value in the **lNegativeCoefficient** member, and the value in **lPositiveCoefficient** is applied

to the entire axis. Likewise, if the effect does not return the DIEFT_POSNEGSATURATION flag, it ignores the value in **dwNegativeSaturation**, and the value in **dwPositiveSaturation** is used as the negative saturation level. Finally, if the effect does not return the DIEFT_SATURATION flag, it ignores both the **dwPositiveSaturation** and **dwNegativeSaturation** values, and no saturation is applied.

You should also check **DIEFFECTINFO.dwEffectType** for the DIEFT_DEADBAND flag, to see if deadband is supported for the condition. If it is not supported, the value in the **lDeadBand** member of the **DICONDITION** structure will be ignored.

You can set a coefficient to a negative value, and this has the effect of generating the force in the opposite direction. For example, for a spring effect, it would cause the spring to push away from the offset point rather than pulling toward it.

[Visual Basic]
To create a condition, pass one of the following values in the *rguid* parameter of the **DirectInputDevice8.CreateEffect** method:

- "GUID_Spring"
- "GUID_Damper"
- "GUID_Inertia"
- "GUID_Friction"

You can also pass any other GUID obtained by the **DirectInputEnumEffects.GetEffectGuid** method, provided the low byte of the value returned by **DirectInputEnumEffects.GetType** is equal to DIEFT_CONDITION. In this way, you can use hardware-specific conditions designed by the manufacturer.

The type-specific parameters for a condition are described in a **DICONDITION** type in the **conditionX** and **conditionY** members of the **DIEFFECT** type.

An application should call the **DirectInputEnumEffects.GetType** method and examine the return value member to determine whether both a positive and a negative coefficient and saturation for the effect are supported on the device. If the effect does not return the DIEFT_POSNEGCOEFFICIENTS flag, it ignores the value in **DICONDITION.lNegativeCoefficient**, and the value in **lPositiveCoefficient** is applied to the entire axis. Likewise, if the effect does not return the DIEFT_POSNEGSATURATION flag, it ignores the value in **lNegativeSaturation** and the value in **lPositiveSaturation** is used as the negative saturation level. Finally, if the effect does not return the DIEFT_SATURATION flag, it ignores both the **lPositiveSaturation** and **lNegativeSaturation** values, and no saturation is applied.

You can set a coefficient to a negative value, and this has the effect of generating the force in the opposite direction. For example, for a spring effect, it would cause the spring to push away from the offset point, rather than pulling toward it.

You should also check the value returned by **GetType** for the DIEFT_DEADBAND flag, to see if deadband is supported for the condition. If it is not supported, the value in the **lDeadBand** member of the **DICONDITION** type will be ignored.

To create a single-axis effect, set the coefficients for the unused axis to 0 in the **conditionX** or **conditionY** member of the **DIEFFECT** type.

DirectX® for Microsoft Visual Basic® does not support rotation of conditions.

Custom Forces

Application writers can create their own effects by creating a custom force. A custom force is an array of constant force values played back by the device.

[C++]
The type-specific structure for custom waveform effects is **DICUSTOMFORCE**.

Set the **dwSamplePeriod** member of the **DICUSTOMFORCE** structure and the **dwSamplePeriod** member of the **DIEFFECT** structure to the same value. This is the length of time, in milliseconds, for which each element in the array of forces is played.

The custom force is played repeatedly until the time set in the **dwDuration** member of the **DIEFFECT** structure has elapsed.

[Visual Basic]
To create a custom force, first define an array of magnitudes, all in the range from –10,000 through 10,000. Then pass this array to the **DirectInputDevice8.CreateCustomEffect** method.

Set the *samplePeriod* parameter and the **lSamplePeriod** member of the **DIEFFECT** type to the same value. This is the length of time, in milliseconds, for which each element in the array of forces is played.

The custom force is played repeatedly until the time set in the **DIEFFECT.lDuration** has elapsed.

Device-Specific Effects

Hardware drivers can support special effects that do not fit into the categories defined by Microsoft® DirectInput®. The type-specific parameters for these effects may be either hard-coded or modifiable by the application.

[C++]

If type-specific parameters are modifiable, the application developer must obtain a header file declaring the data structure required by the effect.

The hardware vendor must provide a GUID identifying the device-specific effect and might provide a custom structure for the type-specific parameters of the effect. Your application then must initialize a **DIEFFECT** structure and a type-specific structure, as with any other effect. You then call the **IDirectInputDevice8::CreateEffect** method, passing the device-specific GUID and a pointer to the **DIEFFECT** structure.

When you obtain information about a device-specific effect in a **DIEFFECTINFO** structure, the low byte of the **dwEffType** member (**DIEFT_GETTYPE**(**dwEfftype**)) indicates into which of the predefined DirectInput effect categories (constant force, ramp force, periodic, or condition) the effect falls. If it does not fall into any of the predefined categories, the value is DIEFT_HARDWARE.

If a device-specific effect falls into one of the predefined categories, the **lpvTypeSpecificParams** member of the **DIEFFECT** structure must point to the corresponding **DICONSTANTFORCE**, **DIRAMPFORCE**, **DIPERIODIC**, or **DICONDITION** structure, and the **cbTypeSpecificParams** member must be equal to the size of that structure.

If (**DIEFT_GETTYPE**(**dwEfftype**) == DIEFT_HARDWARE), the values of the **lpvTypeSpecificParams** and **cbTypeSpecificParams** members depend on whether the effect requires custom type-specific parameters. If it does, these values must refer to the appropriate structure defined in the manufacturer's header file and declared and initialized by your application. If the effect does not require custom parameters—that is, if the **dwStaticParams** member of the **DIEFFECTINFO** structure for the hardware effect does not have the DIEP_TYPESPECIFICPARAMS flag—**lpvTypeSpecificParams** must be NULL and **cbTypeSpecificParams** must be 0.

DirectInput passes the GUID and the **DIEFFECT** structure to the device driver for verification. If the GUID is unknown, the device returns DIERR_DEVICENOTREG. If the GUID is known but the type-specific data is incorrect for that effect, the device returns DIERR_INVALIDPARAM.

[Visual Basic]
Microsoft® DirectX® for Microsoft Visual Basic® does not support hardware effects that require custom type-specific parameters.

The hardware vendor must provide a GUID identifying the device-specific effect. Your application then must initialize a **DIEFFECT** type, as with any other effect. You then pass this type, along with the GUID, to the **DirectInputDevice8.CreateEffect** method.

When you obtain information about a device-specific effect in a **DirectInputEnumEffects** enumeration, the low byte of the return value of **DirectInputEnumEffects.GetType** indicates into which of the predefined DirectInput effect categories (constant force, ramp force, periodic, or condition) the

effect falls. If it does not fall into any of the predefined categories, the value is DIEFT_HARDWARE.

If a device-specific effect falls into one of the predefined categories, the **DIEFFECT** type must be initialized as it would be for a standard force of that category. For example, if the effect type is DIEFT_CONSTANTFORCE, the **constantForce** member must be initialized.

DirectInput passes the GUID and the **DIEFFECT** type to the device driver for verification. If the GUID is unknown, the device returns DIERR_DEVICENOTREG. If the GUID is known but the type-specific data is incorrect for that effect, the device returns DIERR_INVALIDPARAM.

Programming Tips and Tools

This section covers specialized programming topics and an overview of tools provided with the Microsoft® DirectX® SDK that can help in understanding Microsoft DirectInput® data and in designing force-feedback effects.

Information is contained in the following topics.

- Designing for Previous Versions of DirectInput
- Preventing Response Delays During Debugging
- Force Editor

Designing for Previous Versions of DirectInput

[C++]
In the **DirectInput8Create**, **IDirectInput8::Initialize**, **IDirectInputDevice8::Initialize**, and **IDirectInputEffect::Initialize** methods, you are required to pass the Microsoft® DirectInput® version number. This parameter specifies which version of Microsoft DirectX® the DirectInput subsystem should emulate.

Applications designed for the latest version of DirectInput should pass the value DIRECTINPUT_VERSION as defined in Dinput.h.

Applications designed to run under previous versions should pass a value corresponding to the version of DirectInput for which they were designed, with the

main version number in the high-order byte. For example, an application designed to run on DirectInput 3.0 should pass a value of 0x0300.

If you define DIRECTINPUT_VERSION as 0x0300 before including the Dinput.h header file, the header file generates structure definitions compatible with DirectInput 3.0.

If you do not define DIRECTINPUT_VERSION before including the Dinput.h header file, the header file generates structure definitions compatible with the current version of DirectInput. However, the DirectX 3–compatible structures are available under the same names with "_DX3" appended. For example, the DirectX 3–compatible **DIDEVCAPS** structure is called **DIDEVCAPS_DX3**.

You must also use the appropriate versions of the DirectInput and DirectInputDevice interfaces. For versions of DirectX prior to DirectX 7.0, you must use **IDirectInput** and either **IDirectInputDevice** or **IDirectInputDevice2** (the latter is available for DirectX 5.0 or later.)

[Visual Basic]
If you want to design for Microsoft® DirectX® 7 (the only previous version of DirectX that supports Microsoft Visual Basic®), you must include the DirectX 7 for Visual Basic Type Library. Refer to the DirectX 7 documentation for more information.

Preventing Response Delays During Debugging

When using the keyboard or mouse with Microsoft® DirectInput® under Microsoft Windows® 2000, you might notice short delays in response when you are stepping through code during a debugging session. This behavior occurs only in debug mode, so it is not seen by your application's end users. However, this behavior can be eliminated in debug mode as well by adding the Emulation value to the following registry key.

HKEY_LOCAL_MACHINE
  SOFTWARE
    Microsoft
      DirectInput

Name	Type	Data
Emulation	DWORD	0x08 (for the mouse) 0x10 (for the keyboard) 0x18 (for both)

Without changing how DirectInput returns data, this value invokes an emulation mode that is not susceptible to those delays. The Microsoft Software Development Kit (SDK) includes two registry files—Mouse and Keyboard Emulation On.reg and Mouse and Keyboard Emulation Off.reg—to activate and deactivate this emulation mode. The files are located under (*SDK root*)\Samples\Multimedia\DirectInput\Bin.

Note
The Emulation value is supported only in debug mode when your device is using an Exclusive and Foreground cooperative level.

Force Editor

Description

The Force Editor application enables you to design force-feedback effects and test them singly or in combination. Effects can be saved to file and then loaded into Microsoft® DirectInput® applications.

Path

Executable: (*SDK root*)\Bin\Dxutils\Fedit.exe

User's Guide

The Force Editor has its own documentation. Once you've launched Fedit.exe, press F1 for online Help. The Help file can also be launched directly.

(*SDK root)*\Bin\Dxutils\Fedit.chm

DirectInput C/C++ Tutorials

This section contains five tutorials, each providing step-by-step instructions for implementing basic Microsoft® DirectInput® functionality in a C or C++ application.

- Tutorial 1: Using the Keyboard

  The first tutorial shows how to add DirectInput keyboard support to an existing application.
- Tutorial 2: Using the Mouse

  The next tutorial takes you through the steps of providing DirectInput mouse support in an application. The tutorial is based on the Scrawl sample, and focuses on buffered data.
- Tutorial 3: Using the Joystick

This tutorial shows how to enumerate a joystick connected to a system, how to create and initialize a DirectInputDevice object in a callback function, and how to retrieve immediate data. Sample code is based on the JoyStick sample.

- Tutorial 4: Using Force Feedback

  This tutorial illustrates the creation and manipulation of a simple effect on a force-feedback joystick.

- Tutorial 5: Using Action Mapping

  The final tutorial takes you through the creation and application of an action map.

Note

These tutorials are written using C++ style function calls. For instructions on writing Microsoft DirectX® function calls in a C environment, see Using C to Access COM Objects.

Tutorial 1: Using the Keyboard

To prepare for keyboard input, first create an instance of a Microsoft® DirectInput® object. You then use the **IDirectInput8::CreateDevice** method to create an instance of an **IDirectInputDevice8** interface. The **IDirectInputDevice8** interface methods are used to manipulate the device, set its behavior, and retrieve data.

This tutorial divides the required tasks into the following steps.

- Step 1: Creating the DirectInput Object
- Step 2: Creating the DirectInput Keyboard Device
- Step 3: Setting the Keyboard Data Format
- Step 4: Setting the Keyboard Behavior
- Step 5: Gaining Access to the Keyboard
- Step 6: Retrieving Data from the Keyboard
- Step 7: Closing Down the DirectInput System

Adding DirectInput keyboard support to an application is relatively simple, so this tutorial is not based on the complete Keyboard sample application. All of the tutorial steps are illustrated by code fragments within the text. All related steps for initializing the system are gathered in Sample Function 1: DI_Init. Another function, Sample Function 2: DI_Term, is called whenever the system needs to be closed down. Much of the content of these functions are reused in subsequent tutorials.

Step 1: Creating the DirectInput Object

The first step in setting up the Microsoft® DirectInput® system is to create a single DirectInput object as overall manager. This is done with a call to the **DirectInput8Create** function.

```
// HINSTANCE  g_hinst; // initialized earlier
HRESULT        hr;
```

```
LPDIRECTINPUT8  g_lpDI;

hr = DirectInput8Create(g_hinst, DIRECTINPUT_VERSION,
    IID_IDirectInput8, (void**)&g_lpDI, NULL);
if FAILED(hr)
{
   // DirectInput not available; take appropriate action
}
```

The first parameter for **DirectInput8Create** is the instance handle to the application or DLL that is creating the object.

The second parameter tells the DirectInput object which version of the DirectInput system should be used. You can design your application to be compatible with earlier versions of DirectInput. For more information, see Designing for Previous Versions of DirectInput.

The third parameter determines which interface is returned. Most applications obtain the most recent version, by passing IID_IDirectInput8.

The fourth parameter is the address of a variable that is initialized with a valid interface pointer if the call succeeds.

The last parameter specifies the address of the controlling object's **IUnknown** interface for use in COM aggregation. Because most applications do not use aggregation, the value of this parameter is usually NULL.

After creating a single DirectInput object as overall manager, go to Step 2: Creating the DirectInput Keyboard Device.

Step 2: Creating the DirectInput Keyboard Device

After creating the Microsoft® DirectInput® object, your application must create the keyboard object—the device—and retrieve a pointer to an **IDirectInputDevice8** interface. The device performs most of the keyboard-related tasks, using the methods of the interface.

To do this your application must call the **IDirectInput8::CreateDevice** method, as shown in Sample Function 1: DI_Init. **CreateDevice** accepts three parameters.

The first parameter is the GUID for the device being created. Because the system keyboard is used, your application should pass the *GUID_SysKeyboard* predefined global variable.

The second parameter is the address of a variable that is initialized with the interface pointer if the call succeeds.

The third parameter specifies the address of the controlling object's **IUnknown** interface for use in COM aggregation. Because most applications do not use aggregation, the value of this parameter is usually NULL.

The following example attempts to retrieve a pointer to an **IDirectInputDevice8** interface. If this fails, it calls the **DI_Term** application-defined sample function to deallocate existing DirectInput objects, if any.

Note

In all the examples, *g_lpdi* is the initialized pointer to the DirectInput object.

```
HRESULT             hr;
LPDIRECTINPUTDEVICE8  g_lpDIDevice

hr = g_lpDI->CreateDevice(GUID_SysKeyboard, &g_lpDIDevice, NULL);
if FAILED(hr) {
   DI_Term();
   return FALSE;
}
```

After creating the DirectInput keyboard device, go to Step 3: Setting the Keyboard Data Format.

Step 3: Setting the Keyboard Data Format

After retrieving an **IDirectInputDevice8** pointer, your application must set the device's data format, as shown in Sample Function 1: DI_Init. For keyboards, this is a very simple task. Call the **IDirectInputDevice8::SetDataFormat** method, specifying the data format provided by Microsoft® DirectInput® in the *c_dfDIKeyboard* global variable.

The following example attempts to set the data format. If this fails, it calls the **DI_Term** sample function to deallocate existing DirectInput objects, if any.

```
hr = g_lpDIDevice->SetDataFormat(&c_dfDIKeyboard);

if FAILED(hr) {
   DI_Term();
   return FALSE;
}
```

After setting the keyboard data format, go to Step 4: Setting the Keyboard Behavior.

Step 4: Setting the Keyboard Behavior

Before your application can gain access to the keyboard, it must set the device's behavior using the **IDirectInputDevice8::SetCooperativeLevel** method, as shown in Sample Function 1: DI_Init. This method accepts the handle to the window to be associated with the device and a combination of flags that determine the cooperative level. In this case the application is requesting foreground, nonexclusive access.

The following example attempts to set the device's cooperative level. If this fails, it calls the **DI_Term** application-defined sample function to deallocate existing Microsoft® DirectInput® objects, if any.

```
// Set the cooperative level
hr = g_lpDIDevice->SetCooperativeLevel(g_hwndMain,
                DISCL_FOREGROUND | DISCL_NONEXCLUSIVE);

if FAILED(hr) {
   DI_Term();
   return FALSE;
}
```

After setting the keyboard behavior, go to Step 5: Gaining Access to the Keyboard.

Step 5: Gaining Access to the Keyboard

After your application sets the keyboard's behavior, it can acquire access to the device by calling the **IDirectInputDevice8::Acquire** method. The application must acquire the device before retrieving data from it. The **Acquire** method accepts no parameters.

The following line of sample code acquires the keyboard device that was created in Step 2: Creating the DirectInput Keyboard Device.

```
if (g_lpDIDevice) g_lpDIDevice->Acquire();
```

Now that your application has acquired access to the keyboard, go to Step 6: Retrieving Data from the Keyboard.

Step 6: Retrieving Data from the Keyboard

Once a device is acquired, your application can start retrieving data from it. The simplest way to do this is to call the **IDirectInputDevice8::GetDeviceState** method, which takes a snapshot of the device's state at the time of the call.

The **GetDeviceState** method accepts two parameters: the size of a buffer to be filled with device state data, and a pointer to that buffer. For keyboards, always declare a buffer of 256 unsigned bytes.

The following sample attempts to retrieve the state of the keyboard. If this fails, it calls an application-defined sample function to deallocate existing Microsoft® DirectInput® objects, if any. See Sample Function 2: DI_Term.

After retrieving the keyboard's current state, your application can respond to specific keys that were pressed down at the time of the call. Each element in the buffer represents a key. If an element's high bit is 1, the key was down at the moment of the call; otherwise, the key was up. To check the state of a given key, use the DirectInput **Keyboard Device Constants** to index the buffer for a given key.

The following skeleton function, called from the main loop of a hypothetical spaceship game, uses the **IDirectInputDevice8::GetDeviceState** method to poll the keyboard. It then checks to see if the LEFT ARROW, RIGHT ARROW, UP ARROW or DOWN ARROW keys were pressed when the device state was retrieved. This is accomplished with the **KEYDOWN** macro defined in the body of the function. The macro accepts a buffer's variable name and an index value, then checks the byte at the specified index to see if the high bit is set and returns TRUE if it is.

```
void WINAPI ProcessKBInput()
{
    #define KEYDOWN(name, key) (name[key] & 0x80)

    char     buffer[256];
    HRESULT  hr;

    hr = g_lpDIDevice->GetDeviceState(sizeof(buffer),(LPVOID)&buffer);
    if FAILED(hr)
    {
        // If it failed, the device has probably been lost.
        // Check for (hr == DIERR_INPUTLOST)
        // and attempt to reacquire it here.
        return;
    }

    // Turn the spaceship right or left
    if (KEYDOWN(buffer, DIK_RIGHT));
        // Turn right.
    else if(KEYDOWN(buffer, DIK_LEFT));
        // Turn left.

    // Thrust or stop the spaceship
    if (KEYDOWN(buffer, DIK_UP)) ;
        // Move the spaceship forward.
    else if (KEYDOWN(buffer, DIK_DOWN));
        // Stop the spaceship.
}
```

Now that your application can retrieve data from the keyboard, go to **Step 7: Closing Down the DirectInput System**.

Step 7: Closing Down the DirectInput System

When an application is about to close, it should destroy all Microsoft® DirectInput® objects. This is a three-step process.

- Unacquire all DirectInput devices (**IDirectInputDevice8::Unacquire**)
- Release all DirectInput devices (**IDirectInputDevice8::Release**)

- Release the DirectInput object (**IDirectInput8::Release**)

For a sample function that closes down the DirectInput system, see Sample Function 2: DI_Term.

Sample Function 1: DI_Init

This application-defined sample function creates a Microsoft® DirectInput® object, initializes it, and retrieves the necessary interface pointers, assigning them to global variables. When initialization is complete, it acquires the device.

If any part of the initialization fails, this function calls the **DI_Term** application-defined sample function to deallocate DirectInput objects and interface pointers in preparation for terminating the program. See Sample Function 2: DI_Term.

In addition to creating the DirectInput object, the **DI_Init** function performs the tasks discussed in the following tutorial steps.

- Step 2: Creating the DirectInput Keyboard Device
- Step 3: Setting the Keyboard Data Format
- Step 4: Setting the Keyboard Behavior
- Step 5: Gaining Access to the Keyboard

The following is the **DI_Init** function.

```
/* The following variables are presumed to be initialized:
HINSTANCE           g_hinst;    // application instance
HWND                g_hwndMain; // application window
*/

LPDIRECTINPUT8       g_lpDI;
LPDIRECTINPUTDEVICE8  g_lpDIDevice;

BOOL WINAPI DI_Init()
{
    HRESULT hr;

    // Create the DirectInput object.
    hr = DirectInput8Create(g_hinst, DIRECTINPUT_VERSION,
                        IID_IDirectInput8, (void**)&g_lpDI, NULL);

    if FAILED(hr) return FALSE;

    // Retrieve a pointer to an IDirectInputDevice8 interface
    hr = g_lpDI->CreateDevice(GUID_SysKeyboard, &g_lpDIDevice, NULL);

    if FAILED(hr)
```

```
    {
        DI_Term();
        return FALSE;
    }

    // Now that you have an IDirectInputDevice8 interface, get
    // it ready to use.

    // Set the data format using the predefined keyboard data
    // format provided by the DirectInput object for keyboards.

    hr = g_lpDIDevice->SetDataFormat(&c_dfDIKeyboard);

    if FAILED(hr)
    {
        DI_Term();
        return FALSE;
    }

    // Set the cooperative level
    hr = g_lpDIDevice->SetCooperativeLevel(g_hwndMain,
                        DISCL_FOREGROUND | DISCL_NONEXCLUSIVE);
    if FAILED(hr)
    {
        DI_Term();
        return FALSE;
    }

    // Get access to the input device.
    hr = g_lpDIDevice->Acquire();
    if FAILED(hr)
    {
        DI_Term();
        return FALSE;
    }

    return TRUE;
}
```

Sample Function 2: DI_Term

This application-defined sample function deallocates existing Microsoft® DirectInput® interface pointers in preparation for program shutdown or in the event of a failure to properly initialize a device.

```
/* The following variables are presumed initialized:
LPDIRECTINPUT8       g_lpDI;
```

```
LPDIRECTINPUTDEVICE8  g_lpDIDevice;
*/

void WINAPI DI_Term()
{
   if (g_lpDI)
   {
      if (g_lpDIDevice)
      {
      // Always unacquire device before calling Release().
         g_lpDIDevice->Unacquire();
         g_lpDIDevice->Release();
         g_lpDIDevice = NULL;
      }
      g_lpDI->Release();
      g_lpDI = NULL;
   }
}
```

For additional DirectInput tutorials, see DirectInput C/C++ Tutorials.

Tutorial 2: Using the Mouse

This tutorial guides you through the process of setting up a mouse device and retrieving buffered input data. The examples are based on the Scrawl sample.

To prepare for mouse input, you first create an instance of a Microsoft® DirectInput® object. Then you use the **IDirectInput8::CreateDevice** method to create an instance of an **IDirectInputDevice8** interface. The **IDirectInputDevice8** interface methods are used to manipulate the device, set its behavior, and retrieve data.

The preliminary step of setting up the DirectInput system and the final step of closing it down are the same for any application and are covered in Tutorial 1: Using the Keyboard.

This tutorial divides the required tasks into the following steps.

- Step 1: Creating the DirectInput Mouse Device
- Step 2: Setting the Mouse Data Format
- Step 3: Setting the Mouse Behavior
- Step 4: Preparing for Buffered Input from the Mouse
- Step 5: Managing Access to the Mouse
- Step 6: Retrieving Buffered Data from the Mouse

Note

When an application acquires the mouse at the exclusive cooperative level, Microsoft Windows® does not show a mouse pointer on the screen. For this, your application needs a simple sprite engine. The Scrawl sample application uses the Microsoft Win32® function **DrawIcon** to display a crosshair cursor.

Step 1: Creating the DirectInput Mouse Device

After creating the Microsoft® DirectInput® object, your application should retrieve a pointer to an **IDirectInputDevice8** interface, which is used to perform most mouse-related tasks. To do this, call the **IDirectInput8::CreateDevice** method.

The first parameter of **CreateDevice** is the globally unique identifier (GUID) for the device your application is creating. In this case, because the system mouse is used, your application should pass the predefined global variable *GUID_SysMouse*.

The second parameter is the address of a variable that is initialized with a valid **IDirectInputDevice8** interface pointer if the call succeeds.

The third parameter specifies the address of the controlling object's **IUnknown** interface for use in COM aggregation. Because most applications do not use aggregation, the value of this parameter is usually NULL.

The following sample code attempts to retrieve a pointer to an **IDirectInputDevice8** interface. If the call fails, FALSE is returned. It is assumed that *g_pDI* is a valid pointer to **IDirectInput8**.

```
LPDIRECTINPUTDEVICE g_pMouse;
HRESULT             hr;

hr = g_pDI->CreateDevice(GUID_SysMouse, &g_pMouse, NULL);

if (FAILED(hr)) {
   return FALSE;
}
```

After creating the DirectInput mouse device, go to Step 2: Setting the Mouse Data Format.

Step 2: Setting the Mouse Data Format

After retrieving an **IDirectInputDevice8** pointer, your application must set the device's data format. For mouse devices, this is a very simple task. Call the **IDirectInputDevice8::SetDataFormat** method, specifying the data format provided for your convenience by Microsoft® DirectInput® in the *c_dfDIMouse* global variable.

The following code fragment attempts to set the device's data format. If the call fails, FALSE is returned.

```
hr = g_pMouse->SetDataFormat(&c_dfDIMouse);
```

```
if (FAILED(hr)) {
   return FALSE;
}
```

After setting the mouse data format, go to Step 3: Setting the Mouse Behavior.

Step 3: Setting the Mouse Behavior

Before it can gain access to the mouse, your application must set the mouse device's behavior using the **IDirectInputDevice8::SetCooperativeLevel** method. This method accepts the handle to the window to be associated with the device. In Scrawl, the DISCL_EXCLUSIVE flag is included to ensure that this application is the only one that can have exclusive access to the device. This flag is combined with DISCL_FOREGROUND because Scrawl should not allow data input when another application is in the foreground.

The following code sample attempts to set the device's cooperative level. If this attempt fails, FALSE is returned. It is assumed that *hWnd* is a valid window handle.

```
hr = g_pMouse->SetCooperativeLevel(hWnd,
          DISCL_EXCLUSIVE | DISCL_FOREGROUND);

if (FAILED(hr)) {
   return FALSE;
}
```

After setting the mouse behavior, go to Step 4: Preparing for Buffered Input from the Mouse.

Step 4: Preparing for Buffered Input from the Mouse

The Scrawl application demonstrates how to use event notification to find out about mouse activity, and how to read buffered input from the mouse. Both of these techniques require some setup. You can perform these steps at any time after creating the mouse device and before acquiring it.

First, create an event and associate it with the mouse device. You are instructing Microsoft® DirectInput® to notify the mouse device object whenever a hardware interrupt indicates that new data is available.

This is how it is done in Scrawl, where *g_hevtMouse* is a global **HANDLE**.

```
g_hMouseEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (g_hMouseEvent == NULL) {
   return FALSE;
```

```
}

hr = g_pMouse->SetEventNotification(g_hMouseEvent);

if (FAILED(hr)) {
   return FALSE;
}
```

Now set the buffer size so that DirectInput can store any input data until you are ready to look at it. Note that by default the buffer size is zero, so this step is essential if you want to use buffered data.

It is not necessary to used buffered data with event notification. If you prefer, you can retrieve immediate data when an event is signaled.

To set the buffer size, initialize a **DIPROPDWORD** structure with information about itself and about the property you wish to set. Most of the values are boilerplate. The key value is the last one, **dwData**, which is initialized with the number of items you want the buffer to hold.

```
#define SAMPLE_BUFFER_SIZE  16

DIPROPDWORD dipdw;
   // the header
   dipdw.diph.dwSize       = sizeof(DIPROPDWORD);
   dipdw.diph.dwHeaderSize = sizeof(DIPROPHEADER);
   dipdw.diph.dwObj        = 0;
   dipdw.diph.dwHow        = DIPH_DEVICE;
   // the data
   dipdw.dwData            = SAMPLE_BUFFER_SIZE;
```

You then pass the address of the header (the **DIPROPHEADER** structure within the **DIPROPDWORD** structure), along with the identifier of the property you want to change, to the **IDirectInputDevice8::SetProperty** method, as follows:

```
hr = g_pMouse->SetProperty(DIPROP_BUFFERSIZE, &dipdw.diph);

if (FAILED(hr)) {
   return FALSE;
}
```

The setup is now complete, and you are ready to acquire the mouse and start collecting data.

After preparing your application for buffered input from the mouse, go to Step 5: Managing Access to the Mouse.

Step 5: Managing Access to the Mouse

Microsoft® DirectInput® provides the **IDirectInputDevice8::Acquire** and **IDirectInputDevice8::Unacquire** methods to manage device access. Your application must call the **Acquire** method to gain access to the device before requesting mouse information with the **IDirectInputDevice8::GetDeviceState** and **IDirectInputDevice8::GetDeviceData** methods.

Most of the time your application has the device in an acquired state. However, if you have only foreground access, the mouse is automatically unacquired whenever your application moves to the background. You are responsible for reacquiring it when you get the focus back again. This can be done in response to a WM_ACTIVATE message.

Scrawl handles this message by setting a global variable, *g_bActive*, according to whether the application is gaining or losing the focus. It then calls a helper function, **SetAcquire**, which acquires the mouse if *g_bActive* is TRUE and unacquires it otherwise.

```
case WM_ACTIVATE:
    if (WA_INACTIVE == wParam)
        g_bActive = FALSE;
    else
        g_bActive = TRUE;

    // Set exclusive mode access to the mouse based on active state
    SetAcquire();
    return 0;
```

If you have exclusive access, your application might need to unacquire the mouse to enable the user to interact with Microsoft Windows®—for example, to access a menu or a dialog box. In Scrawl this can happen when the user opens the system menu with ALT+SPACEBAR.

The Scrawl window procedure has a handler for WM_ENTERMENULOOP that responds by setting the global variable *g_bActive* to FALSE and calling the **SetAcquire** function. This handler enables Windows to have the mouse and display its own cursor.

When the user is done using a menu, Windows sends the application a WM_EXITSIZEMOVE message. In this case, the Scrawl window process sets *g_bActive* appropriately and calls **SetAcquire**.

Scrawl also unacquires the mouse in response to a right-button click, which displays a shortcut menu. Although the mouse would get unacquired later, in the WM_ENTERMENULOOP handler, it is unacquired here first so that the position of the Windows cursor can be set before the menu appears.

Finally, Scrawl tries to reacquire the mouse if it receives a DIERR_INPUTLOST error after an attempt to retrieve data. This is in case the device has been unacquired

by some mechanism not covered elsewhere; for instance, if the user has pressed CTRL+ALT+DEL.

In summary, your application needs to acquire the mouse before it can get data from it. This needs to be done only once, as long as nothing happens to force your application to surrender access to it. In exclusive mode, you are responsible for giving up control of the mouse when Windows needs it. You are also responsible for reacquiring the mouse whenever your program needs access to it after losing it to Windows or another application.

Once your application can manage access to the mouse, go to Step 6: Retrieving Buffered Data from the Mouse.

Step 6: Retrieving Buffered Data from the Mouse

Once the mouse is acquired, your application can begin to retrieve data from it.

In the Scrawl sample, retrieval is triggered by a signaled event. In the **WinMain** function, the application sleeps until **MsgWaitForMultipleObjects** indicates that there is either a signal or a message. If there is a signal associated with the mouse, the **OnMouseInput** function is called. This function is a good illustration of how buffered input is handled, as shown in the following process.

First, the function makes sure that the old cursor position is cleaned up. Note that Scrawl is maintaining its own cursor and is wholly responsible for drawing and erasing it.

```
VOID OnMouseInput(HWND hWnd)
{
   /* Invalidate the old cursor so that it will be erased */
   InvalidateCursorRect(hWnd);
```

Next, the function enters a loop to read and respond to the entire contents of the buffer. Because it retrieves just one item at a time, it needs only a single **DIDEVICEOBJECTDATA** structure to hold the data.

```
   while (!bDone) {
      DIDEVICEOBJECTDATA od;
      DWORD dwElements = 1;   // number of items to be retrieved
```

Note

Another way to go about handling input would be to read the entire buffer at once and then loop through the retrieved items, responding to each one in turn. In that case, *dwElements* would be the size of the buffer, and *od* would be an array with the same number of elements.

The application calls the **IDirectInputDevice8::GetDeviceData** method in order to fetch the data. The second parameter tells Microsoft® DirectInput® where to put the

data, and the third tells it how many items are wanted. The final parameter would be DIGDD_PEEK if the data was to be left in the buffer, but in this case the data is not needed again, and so it is removed.

```
hr = g_pMouse->GetDeviceData(sizeof(DIDEVICEOBJECTDATA),
                    &od, &dwElements, 0);
```

Next, the application checks to see if access to the device has been lost and, if so, attempts to reacquire the mouse at the first opportunity. This step was discussed in Step 5: Managing Access to the Mouse.

```
if (hr == DIERR_INPUTLOST)
{
   SetAcquire();
   break;
}
```

Next, the application makes sure the call to the **GetDeviceData** method succeeded and that there was actually data to be retrieved. Note that after the call to **GetDeviceData** the *dwElements* variable shows how many items were actually retrieved.

```
/* Unable to read data or no data available */
if (FAILED(hr) || dwElements == 0)
{
   break;
}
```

If execution has proceeded to this point, the call succeeded and there is an item of data in the buffer. Now the application looks at the **dwOfs** member of the **DIDEVICEOBJECTDATA** structure to determine which object on the device reported a change of state, and it calls helper functions to respond appropriately. The value of the **dwData** member, which contains information about what occurred with the device object, is passed to these functions.

```
/* Look at the element to see what occurred */

switch (od.dwOfs)
{
   // Mouse horizontal motion
   case DIMOFS_X:
      UpdateCursorPosition(od.dwData, 0);
      break;

   // Mouse vertical motion
   case DIMOFS_Y:
      UpdateCursorPosition(0, od.dwData);
      break;
```

```
        // DIMOFS_BUTTON0: Right button pressed or released
        case DIMOFS_BUTTON0:
        // DIMOFS_BUTTON1: Left button pressed or released
        case DIMOFS_BUTTON1:
          // Is the right button or a swapped left button down?
          if((g_bSwapMouseButtons &&
                    DIMOFS_BUTTON1 == od.dwOfs) ||
            (!g_bSwapMouseButtons &&
                    DIMOFS_BUTTON0 == od.dwOfs))
          {
            if (od.dwData & 0x80)  // Left button pressed, so
                              // go into button-down mode
            {
              bDone = TRUE;
              OnLeftButtonDown(hWnd);
            }
          // Is the left button or a swapped right button down?
          if((g_bSwapMouseButtons &&
                    DIMOFS_BUTTON0 == od.dwOfs) ||
            (!g_bSwapMouseButtons &&
                    DIMOFS_BUTTON1 == od.dwOfs))
          {
            if(!(od.dwData & 0x80))  // button release, so
                               // check shortcut menu
            {
              bDone = TRUE;
              OnRightButtonUp(hWnd);
            }
          }
          break;
      }
```

Finally, in the event that the cursor has been moved by one of the helper functions, the **OnMouseInput** sample function invalidates the screen rectangle occupied by the cursor.

```
    // Invalidate the new cursor so that it will be drawn
    InvalidateCursorRect(hWnd);
}
```

Scrawl also collects mouse data in the **OnLeftButtonDown** function. This is where the application keeps track of mouse movements while the primary button is being held down—that is, while the user is drawing. This function does not rely on event notification, but repeatedly polls the DirectInput buffer until the button is released.

Note that in the **OnLeftButtonDown** function no drawing is done until all pending data has been read. This is because each horizontal or vertical movement of the mouse is reported as a separate event. (Both events are, however, placed in the buffer at the same time.) If a line were immediately drawn in response to each separate axis movement, a diagonal mouse movement would produce two lines at right angles.

Another way you can be sure that the movement in both axes is taken into account before responding in your application is to check the sequence numbers of the x-axis item and the y-axis item. If the numbers are the same, the two events took place simultaneously. For more information, see Time Stamps and Sequence Numbers.

For additional DirectInput tutorials, see DirectInput C/C++ Tutorials.

Tutorial 3: Using the Joystick

This tutorial shows you how to enumerate a joystick attached to a system and set it up for input. Code samples are based on the JoyStick sample.

The preliminary step of setting up the Microsoft® DirectInput® system and the final step of closing it down are the same for any application and are covered in Tutorial 1: Using the Keyboard.

The first step in this tutorial is to enumerate devices; that is, to see what joysticks are available. As part of this process you initialize each joystick device and set its desired characteristics. You then use the **IDirectInputDevice8** interface methods to retrieve data from each joystick.

This tutorial divides the required tasks into the following steps.

- Step 1: Enumerating the Joysticks
- Step 2: Creating the DirectInput Joystick Device
- Step 3: Setting the Joystick Data Format
- Step 4: Setting the Joystick Behavior
- Step 5: Gaining Access to the Joystick
- Step 6: Retrieving Data from the Joystick

Step 1: Enumerating the Joysticks

After creating the Microsoft® DirectInput® system, call the **IDirectInput8::EnumDevices** method to enumerate the joysticks, as in the following sample code.

```
//g_pDI is an initialized pointer to IDirectInput8

g_pDI->EnumDevices(DI8DEVCLASS_GAMECTRL, EnumJoysticksCallback,
                   NULL, DIEDFL_ATTACHEDONLY)
```

The DI8DEVCLASS_GAMECTRL constant, passed as the first parameter, specifies the class of device to be enumerated.

EnumJoysticksCallback is the address of a callback function to be called each time a joystick is found. This is where the individual devices are created in Step 2: Creating the DirectInputJoystick Device.

The third parameter can be any 32-bit value that you want to make available to the callback function. In this case, nothing is being sent.

The last parameter, DIEDFL_ATTACHEDONLY, is a flag that restricts enumeration to devices that are attached to the computer.

When your application can enumerate the joysticks, go to Step 2: Creating the DirectInput Joystick Device.

Step 2: Creating the DirectInput Joystick Device

After creating the Microsoft® DirectInput® object, the application must retrieve a pointer to an **IDirectInputDevice8** interface, which is used to perform most joystick-related tasks. In the JoyStick sample, this is done in the callback function **EnumJoysticksCallback**, which is called each time a joystick is enumerated.

The following sample code is the first part of the callback function.

```
BOOL CALLBACK EnumJoysticksCallback(const DIDEVICEINSTANCE*
                                    pdidInstance, VOID* pContext)
{
    HRESULT hr;

    // Obtain an interface to the enumerated joystick.
    hr = g_pDI->CreateDevice(pdidInstance->guidInstance,
                             &g_pJoystick, NULL);
    if(FAILED(hr))
        return DIENUM_CONTINUE;

    return DIENUM_STOP;
```

The **EnumJoysticksCallback** callback function parameters are as follows:

- A pointer to the device instance, supplied by the DirectInput system when the device is enumerated.
- A pointer to a 32-bit value that you supplied as a parameter to **IDirectInput8::EnumDevices**. This parameter could be any 32-bit value such as a pointer to an interface so that you could use its methods. In this case, however, NULL was passed.

The first task of the callback function is to create the device. The **IDirectInput8::CreateDevice** method accepts three parameters.

The first parameter is a reference to the globally unique identifier (GUID) for the instance of the device. In this case, the GUID is taken from the **DIDEVICEINSTANCE** structure supplied by DirectInput when it enumerated the device.

The second parameter is the address of the variable that is initialized with a valid interface pointer if the call succeeds.

The third parameter specifies the address of the controlling object's **IUnknown** interface for use in COM aggregation. The JoyStick sample does not use aggregation, so the parameter is NULL.

Note that if the device interface cannot be created, DIENUM_CONTINUE is returned from the callback function. This flag instructs DirectInput to keep enumerating as long as there are devices to be enumerated.

After creating the DirectInput joystick device, go to Step 3: Setting the Joystick Data Format.

Step 3: Setting the Joystick Data Format

Now that the application has a pointer to a Microsoft® DirectInput® device, it can call the **IDirectInputDevice8** methods to manipulate that device. The first step, which is essential, is to set the data format for the joystick. This step tells DirectInput how to format the input data.

```
if (FAILED(hr = g_pJoystick->SetDataFormat(&c_dfDIJoystick2)))
    return hr;
```

The **IDirectInputDevice8::SetDataFormat** method takes one parameter, a pointer to a **DIDATAFORMAT** structure containing information about how the data for the device is to be formatted. For the joystick, you can use the predefined global variable *c_dfDIJoystick2*, which signifies use of the **DIJOYSTATE2** structure to retrieve data.

After setting the joystick data format, go to Step 4: Setting the Joystick Behavior.

Step 4: Setting the Joystick Behavior

The joystick device has been created, and its data format has been set. The next step is to set its cooperative level. As in the previous step, *g_pJoystick* is a pointer to the device interface.

```
if (FAILED(hr = g_pJoystick->SetCooperativeLevel(hDlg,
                DISCL_EXCLUSIVE | DISCL_FOREGROUND)))
    return hr;
```

The first parameter to **IDirectInputDevice8::SetCooperativeLevel** is a window handle. In this case, the handle to the dialog box serving as the main program window is passed to the function.

The final parameter is a combination of flags describing the desired cooperative level. The Joystick sample requires input from the joystick only when it is the foreground application. Also, when the joystick sample is in the foreground, it should be the only application with exclusive access to the joystick. Therefore, the flags are set to DISCL_EXCLUSIVE | DISCL_FOREGROUND. See Cooperative Levels for a full explanation of these flags.

The next step is to gather information about the capabilities of the joystick. This information includes the number of axes, buttons, and other controllers, whether the device supports force feedback, and other details about the joystick's behavior and supported options.

A **DIDEVCAPS** structure is used to hold the capability information. The **dwSize** member of the **DIDEVCAPS** structure must first be initialized, then the address of the structure sent to **IDirectInputDevice8::GetCapabilities**. In the example below, *g_diDevCaps* is a previously declared **DIDEVCAPS** structure.

```
g_diDevCaps.dwSize = sizeof(DIDEVCAPS);
if (FAILED(hr = g_pJoystick->GetCapabilities(&g_diDevCaps)))
    return hr;
```

At this point, you could proceed using the device defaults or you could customize the device properties. The Joystick sample enumerate the axes on the joystick and sets a range for each by calling **IDirectInputDevice8::EnumObjects**.

```
if (FAILED(hr = g_pJoystick->EnumObjects(EnumAxesCallback,
                                         (VOID*)hDlg, DIDFT_AXIS)))
    return hr;
```

The first parameter to the **EnumObjects** method, *EnumAxesCallback,* is the address of a callback function which processes the enumerated objects as desired.

The second parameter can be any 32-bit value that you might want to provide to the callback function. Here, the handle to the main dialog window is passed so that it can update the display to indicate which axes are found on the joystick.

The last parameter, DIDFT_AXIS, tells the callback function to enumerate only device axes.

In the sample, the properties changed will be the range for any axis or slider detected (a slider is considered an axis).

By setting the range, you are telling DirectInput what maximum and minimum values you want returned for an axis. If you set a range of -1,000 to +1,000 for the x-axis, as in the example, you are asking that a value of -1,000 be returned when the stick is at the extreme left, +1,000 when it is at the extreme right, and zero when it is in the middle.

The following sample code sets the range of an enumerated axis. The *pdidoi* variable is the address of a **DIDEVICEOBJECTINSTANCE** structure passed by the system to the callback function. This variable contains information about the object currently being enumerated.

```
DIPROPRANGE diprg;

diprg.diph.dwSize       = sizeof(DIPROPRANGE);
diprg.diph.dwHeaderSize = sizeof(DIPROPHEADER);
diprg.diph.dwHow        = DIPH_BYID;
diprg.diph.dwObj        = pdidoi->dwType;
diprg.lMin              = -1000;
diprg.lMax              = +1000;
.
.
hr = g_pJoystick->SetProperty(DIPROP_RANGE, &diprg.diph);
if (FAILED(hr))
   return DIENUM_STOP;
```

The first task here is to set up the **DIPROPRANGE** structure *diprg*, whose address is passed into the **IDirectInputDevice8::SetProperty** method. Note that it is not the address of the structure itself that is passed but rather the address of its first member, which is a **DIPROPHEADER** structure. For more information, see Device Properties.

The property header is initialized with the following values.

- The size of the property structure
- The size of the header structure
- A flag to indicate how the **dwObj** member is to be interpreted.
- The value found in the **dwType** member of the **DIDEVICEOBJECTINSTANCE** structure indicating the object whose property is being changed.

The **lmin** and **lmax** members of the **DIPROPRANGE** structure are assigned the desired range values.

The application now calls the **IDirectInputDevice8::SetProperty** method. The first parameter is a flag indicating which property is being changed. The second parameter is the address of the **DIPROPHEADER** member of the property structure.

After setting the joystick behavior, go to Step 5: Gaining Access to the Joystick.

Step 5: Gaining Access to the Joystick

After your application sets a joystick's behavior, it can acquire access to the device by calling the **IDirectInputDevice8::Acquire** method. The application must acquire the device before retrieving data from it. The **Acquire** method accepts no parameters.

The Joystick sample performs initial acquisition in the message loop as the application is activated. This loop also automatically reacquires the device whenever the application regains focus after having lost it. Acquisition is also performed if access to the joystick is lost or a DIERR_INPUTLOST error is returned when the application attempts to get input data. This is shown in the following code sample.

```
hr = g_pJoystick->Poll();
if (FAILED(hr))
{
    hr = g_pJoystick->Acquire();
    while(hr == DIERR_INPUTLOST)
        hr = g_pJoystick->Acquire();
    return S_OK;
}
```

Once your application can gain access to the joystick go to Step 6: Retrieving Data from the Joystick.

Step 6: Retrieving Data from the Joystick

Because your application is more likely concerned with the position of the joystick axes than with their movement, you probably want to retrieve immediate rather than buffered data from the device. You can do this by polling with **IDirectInputDevice8::GetDeviceState**. Note that not all device drivers notify Microsoft® DirectInput® when the state of the device changes. Therefore, it is always good policy to call the **IDirectInputDevice8::Poll** method before checking the device state.

The Joystick application calls the following function from the Microsoft® Windows® message loop.

```
HRESULT UpdateInputState(HWND hDlg)
{
    HRESULT     hr;
    CHAR        strText[128]; // Device state text
    DIJOYSTATE2 js;           // Direct Input joystick state
    CHAR*       str;

    if (NULL == g_pJoystick)
        return S_OK;

    // Poll the device to read the current state
    hr = g_pJoystick->Poll();
    if (FAILED(hr))
    {
        // Attempt to reacquire joystick
    }
    hr = g_pJoystick->GetDeviceState(sizeof(DIJOYSTATE2), &js);
```

```
if (FAILED(hr))
    return hr;
```

Note the call to **GetDeviceState**. The first parameter is the size of the structure in which the data is returned, and the second parameter is the address of this structure, which is of type **DIJOYSTATE2**. This structure holds data for up to six axes, 128 buttons, and four point-of-view hats. The sample program proceeds to look at the state of all axes and buttons, and one slider. This information is then displayed in the main dialog window.

Joystick buttons work just like keys or mouse buttons. If the high bit of the returned byte is 1, the button is pressed.

For additional DirectInput tutorials, see DirectInput C/C++ Tutorials.

Tutorial 4: Using Force Feedback

This tutorial takes you through the process of creating, playing, and modifying a simple effect on a force-feedback joystick. The effect is based on the feel of running a chain saw. Effects are created that simulate a chain saw starting with some difficulty, running normally, and cutting into wood.

The preliminary step of setting up the Microsoft® DirectInput® system and the final step of closing it down are essentially the same for any application and are covered in Tutorial 1: Using the Keyboard. However, when closing down the DirectInput force-feedback system, you are required to take the additional step of releasing any effects you have created.

This tutorial divides the required tasks into the following steps.

- Step 1: Enumerating Force-Feedback Devices
- Step 2: Creating the DirectInput Force-Feedback Device
- Step 3: Enumerating Supported Effects
- Step 4: Creating an Effect
- Step 5: Playing an Infinite Effect
- Step 6: Changing an Effect

Step 1: Enumerating Force-Feedback Devices

The first step is to ensure that a force-feedback device is available on the system. You do this by calling the **IDirectInput8::EnumDevices** method. In the following example, the global pointer to the game device interface is initialized only if the enumeration has succeeded in finding at least one suitable device:

```
LPDIRECTINPUTDEVICE8  g_lpDIDevice = NULL;

hr = g_lpDI->EnumDevices(DI8DEVTYPE_GAMECTRL,
```

```
                DIEnumDevicesProc,
                NULL,
                DIEDFL_FORCEFEEDBACK | DIEDFL_ATTACHEDONLY);
if (FAILED(hr))
{
  // No force-feedback joystick available; take appropriate action.
}
```

In the example, *g_lpDI* is an initialized pointer to the **IDirectInput8** interface. The first parameter to **EnumDevices** restricts the enumeration to joystick-type devices. The second parameter is the callback function that is called whenever Microsoft® DirectInput® identifies a device that qualifies for enumeration. The third parameter is for user-defined data to be passed in or out of the callback function; in this case it is not used. Finally, the flags further restrict the enumeration to devices attached to the system that support force feedback.

The callback function is a convenient place to initialize the device as soon as it has been found. (It is assumed that the first device found is the one you want to use.) You do this in Step 2: Creating the DirectInput Force-Feedback Device.

Step 2: Creating the DirectInput Force-Feedback Device

To have Microsoft® DirectInput® enumerate devices, create a callback function of the same type as **DIEnumDevicesCallback**. In Step 1 you passed the address of this function to the **IDirectInput8::EnumDevices** method.

DirectInput passes into the callback, as the first parameter, a pointer to a **DIDEVICEINSTANCE** structure that tells you what you need to know about the device. The structure member of most interest in the example is **guidInstance**, the unique identifier for the particular piece of hardware on the user's system. You must pass this GUID to the **IDirectInput8::CreateDevice** method.

The following code fragment is the first part of the callback, which extracts the GUID and creates the device object.

```
BOOL CALLBACK DIEnumDevicesProc( LPCDIDEVICEINSTANCE lpddi,
                        LPVOID pvRef )
{
    HRESULT hr;
    GUID DeviceGuid = lpddi->guidInstance;

    // Create game device.

    hr = g_lpDI->CreateDevice(DeviceGuid, &g_lpDIDevice, NULL);

    if (FAILED(hr))
        // Continue enumerating until successful or finished
```

```
    return DIENUM_CONTINUE;
```

The next steps of the callback function are similar to those for setting up any input device. Note that you need the exclusive cooperative level for any force-feedback device.

```
    // Set cooperative level.
    hr = g_lpDIDevice->SetCooperativeLevel(g_hwndMain,
                        DISCL_EXCLUSIVE | DISCL_FOREGROUND);
    if (FAILED(hr))
        return hr;

    // set game data format
    hr = g_lpDIDevice->SetDataFormat(&c_dfDIJoystick);
    if (FAILED(hr))
        return hr;
```

Finally, you might want to turn off the device's autocenter feature. Autocenter is essentially a condition effect that uses the motors to simulate the springs in a standard joystick. It is a good idea to turn it off so that it does not interfere with other effects.

```
    DIPROPDWORD DIPropAutoCenter;

    DIPropAutoCenter.diph.dwSize       = sizeof(DIPropAutoCenter);
    DIPropAutoCenter.diph.dwHeaderSize = sizeof(DIPROPHEADER);
    DIPropAutoCenter.diph.dwObj        = 0;
    DIPropAutoCenter.diph.dwHow        = DIPH_DEVICE;
    DIPropAutoCenter.dwData            = DIPROPAUTOCENTER_OFF;

    hr = g_lpDIDevice->SetProperty(DIPROP_AUTOCENTER,
                        &DIPropAutoCenter.diph);
    if (FAILED(hr))
    {
        // Handle the failure as appropriate
    }

    return DIENUM_STOP;   // One is enough.
}   // end DIEnumDevicesProc
```

Before using the device, you must acquire it. See Step 5: Gaining Access to the Joystick in Tutorial 3: Using the Joystick for an example of how to handle acquisition. Note that unlike other property changes, force-feedback effects can be downloaded to a device when it is in an acquired state.

After you have created the DirectInput force-feedback device, go to Step 3: Enumerating Supported Effects.

Step 3: Enumerating Supported Effects

Now that you have successfully enumerated and created a force-feedback device, you can enumerate the effect types it supports.

Effect enumeration is not strictly necessary if you want to create only standard effects that will be available on any device, such as constant forces. When creating the effect object, you can identify the desired effect type simply by using one of the predefined GUIDs, such as *GUID_ConstantForce*. For a complete list of these identifiers, see **IDirectInputDevice8::CreateEffect**.

Another more flexible approach is to enumerate supported effects of a particular type, then obtain the GUID for the effect from the callback function. You could use the callback to obtain more information about the device's support for the effect— for example, whether it supports an envelope—but in this tutorial you obtain only the effect GUID.

First, create the callback function that Microsoft® DirectInput® calls for each effect enumerated. For information on this standard callback, see **DIEnumEffectsCallback**. You can give the function any name you like.

```
BOOL  EffectFound = FALSE;  // global flag

BOOL CALLBACK DIEnumEffectsProc(LPCDIEFFECTINFO pei, LPVOID pv)
{
   *((GUID *)pv) = pei->guid;
   EffectFound = TRUE;
   return DIENUM_STOP;  // one is enough
}
```

The GUID variable pointed to by the application-defined value *pv* is assigned the value passed in the **DIEFFECTINFO** structure created by DirectInput for the effect.

To obtain the effect GUID, set the callback in motion by calling the **IDirectInputDevice8::EnumEffects** method, as follows:

```
HRESULT  hr;
GUID     guidEffect;

hr = g_lpDIDevice->EnumEffects(
                    (LPDIENUMEFFECTSCALLBACK) DIEnumEffectsProc,
                     &guidEffect,
                     DIEFT_PERIODIC);
if (FAILED(hr))
{
   // Note that success does not mean that any effects were found,
   // only that the process went smoothly.
}
```

Note that you pass the address of a GUID variable, *guidEffect*, to the **EnumEffects** method. This address is passed in turn to the callback as the *pv* parameter. You also restrict the enumeration to periodic effects by setting the flag DIEFT_PERIODIC.

Once your application can enumerate supported effects, go to Step 4: Creating an Effect.

Step 4: Creating an Effect

If the *EffectFound* value is no longer FALSE after effect enumeration, you can safely assume that Microsoft® DirectInput® has found support for at least one effect of the periodic type you requested by passing the DIEFT_PERIODIC flag. Knowing the effect GUID, you can now create the effect object.

Before calling the **IDirectInputDevice8::CreateEffect** method, you set up the following arrays and structures.

- An array of axes that will be involved in the effect. For a joystick, this array normally consists of the identifiers for the x-axis and the y-axis.
- An array of values for setting the direction. The values differ according to both the number of axes and whether you want to use polar, spherical, or Cartesian coordinates. For a full explanation, see Effect Direction.
- A structure of type-specific parameters. In the example, because you are creating a periodic effect, this is of type **DIPERIODIC**.
- A **DIENVELOPE** structure for defining an envelope to be applied to the effect.
- A **DIEFFECT** structure to contain the basic parameters for the effect.

Arrays can be initialized when they, along with the structures, are declared.

```
DWORD      dwAxes[2] = {DIJOFS_X, DIJOFS_Y};
LONG       lDirection[2] = {0, 0};

DIPERIODIC diPeriodic;      // type-specific parameters
DIENVELOPE diEnvelope;      // envelope
DIEFFECT   diEffect;        // general parameters
```

Next, initialize the type-specific parameters. The values in this example create a full-force periodic effect with a period of one-twentieth of a second.

```
diPeriodic.dwMagnitude = DI_FFNOMINALMAX;
diPeriodic.lOffset = 0;
diPeriodic.dwPhase = 0;
diPeriodic.dwPeriod = (DWORD)(0.05 * DI_SECONDS);
```

To produce an effect of the chain-saw motor trying to start, briefly coughing into life, and then slowly dying, you set an envelope with an attack time of one-half second and a fade time of one second.

```
diEnvelope.dwSize = sizeof(DIENVELOPE);
diEnvelope.dwAttackLevel = 0;
diEnvelope.dwAttackTime = (DWORD)(0.5 * DI_SECONDS);
diEnvelope.dwFadeLevel = 0;
diEnvelope.dwFadeTime = (DWORD)(1.0 * DI_SECONDS);
```

Next, set up the basic effect parameters. These include flags to determine how the directions and device objects (buttons and axes) are identified, the sample period and gain for the effect, and pointers to the other data that you have just prepared. You also associate the effect with the joystick's fire button so that it is automatically played whenever that button is pressed.

```
diEffect.dwSize = sizeof(DIEFFECT);
diEffect.dwFlags = DIEFF_POLAR | DIEFF_OBJECTOFFSETS;
diEffect.dwDuration = (DWORD)(2 * DI_SECONDS);

diEffect.dwSamplePeriod = 0;              // = default
diEffect.dwGain = DI_FFNOMINALMAX;        // no scaling
diEffect.dwTriggerButton = DIJOFS_BUTTON0;
diEffect.dwTriggerRepeatInterval = 0;
diEffect.cAxes = 2;
diEffect.rgdwAxes = dwAxes;
diEffect.rglDirection = &lDirection[0];
diEffect.lpEnvelope = &diEnvelope;
diEffect.cbTypeSpecificParams = sizeof(diPeriodic);
diEffect.lpvTypeSpecificParams = &diPeriodic;
```

Finally, you can create the effect.

```
LPDIRECTINPUTEFFECT  g_lpdiEffect;  // global effect object

hr = g_lpDIDevice->CreateEffect(
                guidEffect,     // GUID from enumeration
                &diEffect,      // where the data is
                &g_lpdiEffect,  // where to put interface pointer
                NULL);          // no aggregation
if (FAILED(hr))
{
   return hr;
}
```

Note that, by default, the effect is downloaded to the device as soon as it has been created, provided that the device is in an acquired state at the exclusive cooperative level. You should now be able to compile, run, press the fire button, and feel the sputtering of a chain saw that's out of gas.

After creating an effect object, go to Step 5: Playing an Infinite Effect.

Step 5: Playing an Infinite Effect

The effect created in the previous step starts in response to the press of a button. To create an effect that is to be played in response to an explicit call, return to Step 4: Creating an Effect and modify the **dwTriggerButton** member of the **DIEFFECT** structure as follows:

```
diEffect.dwTriggerButton = DIEB_NOTRIGGER;
```

To make a chain saw that starts and keeps going, change the **dwDuration** member as follows:

```
diEffect.dwDuration = INFINITE;
```

Next, start the effect.

```
g_lpdiEffect->Start(1, 0);
```

The effect keeps running until you stop it.

```
g_lpdiEffect->Stop();
```

Note that it is not necessary to change the envelope you created in the previous step. The attack is played as the effect starts, but the fade value is ignored.

Once your application can play an infinite effect, go to Step 6: Changing an Effect.

Step 6: Changing an Effect

Next, you modify the effect to simulate the slowing down of the engine as the saw bites into wood. Microsoft® DirectInput® enables you to modify the parameters of an effect while it is playing.

To change the effect, set up a new **DIEFFECT** structure or have access to the one you used to create the effect. If you are setting up a new structure with local scope, initialize only the **dwSize** member and any members that contain or point to data that is to be changed.

In this case, you want to change a type-specific parameter—the period of the effect—so you need to have access to the **DIPERIODIC** structure you used when creating the effect. Alternatively, you can create a local copy with all members initialized. Make sure that the address of the **DIPERIODIC** structure is in the **lpvTypeSpecificParams** member of the **DIEFFECT** structure.

First, set the new period of the effect.

```
diPeriodic.dwPeriod = (DWORD)(0.08 * DI_SECONDS);
```

Next, call the **IDirectInputEffect::SetParameters** method to make the changes.

```
hr = g_lpdiEffect->SetParameters(&diEffect, DIEP_TYPESPECIFICPARAMS);
```

Note the DIEP_TYPESPECIFICPARAMS flag that restricts the changes to a single member of the **DIEFFECT** structure.

You can control the way changes are handled by using other flags. For example, by using the DIEP_NODOWNLOAD flag you could change the parameters immediately after starting the effect but delay the implementation until a specific call to the **IDirectInputEffect::Download** method. For more information concerning use of the various control flags, see **IDirectInputEffect::SetParameters**.

For additional DirectInput tutorials, see DirectInput C/C++ Tutorials.

Tutorial 5: Using Action Mapping

This tutorial guides you through the setup, modification, and application of an action map. Examples are based written as though they were to be used in a space combat game.

The preliminary step of setting up the Microsoft® DirectInput® system and the final step of closing it down are the same for any application and are covered in Tutorial 1: Using the Keyboard. The keyboard, a mouse, and a joystick are all used in the sample, so each of those devices needs to be set up.

This tutorial divides the required tasks into the following steps.

- Step 1: Defining the Game Actions
- Step 2: Defining the Action Map
- Step 3: Mapping Actions to Devices
- Step 4: Configuring and Applying the Action Map
- Step 5: Displaying the Action Map
- Step 6: User Configuration of the Action Map
- Step 7: Retrieval of Action Mapped Data

Step 1: Defining the Game Actions

The first step toward creating an action map is the definition of the game actions. These actions are specific to the application and are defined by the developer. In the case of this example, the actions chosen reflect its needs as a space combat simulator. These values can be defined in many ways as the developer sees fit. They could be defined as an enumeration, as in the Preparing the Action Map overview, or they could be defined as constants as in this code fragment.

```
#define INPUT_LEFTRIGHT_AXIS   1L
#define INPUT_UPDOWN_AXIS      2L
#define INPUT_TURNLEFT         3L
#define INPUT_TURNRIGHT        4L
#define INPUT_FORWARDTHRUST    5L
```

```
#define INPUT_REVERSETHRUST    6L
#define INPUT_FIREWEAPONS      7L
#define INPUT_ENABLESHIELD     8L
#define INPUT_DISPLAYGAMEMENU  9L
#define INPUT_QUITGAME         10L
```

Once these values are defined, they can be assigned to genre-specific action-mapping constants. This is done in Step 2: Defining the Action Map.

Step 2: Defining the Action Map

Next, each game action should be assigned to at least one virtual control or device object. These controls are defined as action-mapping constants and are defined by genre and subgenre. In this case, the best choice is the Space Combat subgenre of the flight-games genre. This subgenre makes the following constants available.

Priority 1 Controls

DIAXIS_SPACESIM_LATERAL
DIAXIS_SPACESIM_MOVE
DIAXIS_SPACESIM_THROTTLE
DIBUTTON_SPACESIM_FIRE
DIBUTTON_SPACESIM_MENU
DIBUTTON_SPACESIM_TARGET
DIBUTTON_SPACESIM_WEAPONS

Priority 2 Controls

DIAXIS_SPACESIM_CLIMB
DIAXIS_SPACESIM_ROTATE
DIBUTTON_SPACESIM_BACKWARD_LINK
DIBUTTON_SPACESIM_DEVICE DIBUTTON_SPACESIM_DISPLAY
DIBUTTON_SPACESIM_FASTER_LINK
DIBUTTON_SPACESIM_FIRESECONDARY
DIBUTTON_SPACESIM_FORWARD_LINK
DIBUTTON_SPACESIM_GEAR
DIBUTTON_SPACESIM_GLANCE_DOWN_LINK
DIBUTTON_SPACESIM_GLANCE_LEFT_LINK
DIBUTTON_SPACESIM_GLANCE_RIGHT_LINK
DIBUTTON_SPACESIM_GLANCE_UP_LINK
DIBUTTON_SPACESIM_LEFT_LINK DIBUTTON_SPACESIM_LOWER
DIBUTTON_SPACESIM_PAUSE DIBUTTON_SPACESIM_RAISE
DIBUTTON_SPACESIM_RIGHT_LINK
DIBUTTON_SPACESIM_SLOWER_LINK
DIBUTTON_SPACESIM_TURN_LEFT_LINK
DIBUTTON_SPACESIM_TURN_RIGHT_LINK
DIBUTTON_SPACESIM_VIEW
DIHATSWITCH_SPACESIM_GLANCE

Priority 1 controls are those controls that, at the minimum, should be mapped to a device object if possible. Priority 2 controls are less critical to basic game operation and may be mapped as needed.

Game actions are assigned to action-mapping constant controls in an array of DIACTION structures. Each **DIACTION** structure contains at least one game action, its associated action-mapping constant, any necessary flags, and either a friendly name or ID string for the action. The remaining **DIACTION** members are not used at this point. The following code sample shows definition of the array of **DIACTION** structures.

```
#define NUMBER_OF_ACTIONS 18

DIACTION g_rgGameAction[NUMBER_OF_ACTIONS] =
{
    // Device input pre-defined by DInput, according to genre
    {INPUT_LEFTRIGHT_AXIS, DIAXIS_SPACESIM_LATERAL, 0, "Turn",},
    {INPUT_UPDOWN_AXIS, DIAXIS_SPACESIM_MOVE, 0, "Move",},
    {INPUT_FIREWEAPONS, DIBUTTON_SPACESIM_FIRE, 0, "Shoot",},
    {INPUT_ENABLESHIELD, DIBUTTON_SPACESIM_GEAR, 0, "Shield",},
    {INPUT_DISPLAYGAMEMENU, DIBUTTON_SPACESIM_DISPLAY, 0, "Display",},
    {INPUT_QUITGAME, DIBUTTON_SPACESIM_MENU, 0, "Quit Game",},

    // Keyboard input mappings
    {INPUT_TURNLEFT, DIKEYBOARD_LEFT, 0, "Turn left", },
    {INPUT_TURNRIGHT, DIKEYBOARD_RIGHT, 0, "Turn right", },
    {INPUT_FORWARDTHRUST, DIKEYBOARD_UP, 0, "Forward thrust", },
    {INPUT_REVERSETHRUST, DIKEYBOARD_DOWN, 0, "Reverse thrust", },
    {INPUT_FIREWEAPONS, DIKEYBOARD_F, 0, "Fire weapons", },
    {INPUT_ENABLESHIELD, DIKEYBOARD_S, 0, "Enable shields", },
    {INPUT_DISPLAYGAMEMENU, DIKEYBOARD_D, DIA_APPFIXED,
                        "Display game menu"), },
    {INPUT_QUITGAME, DIKEYBOARD_ESCAPE, DIA_APPFIXED, "Quit game", },

    // Mouse input mappings
    {INPUT_LEFTRIGHT_AXIS, DIMOUSE_XAXIS, 0, "Turn", },
    {INPUT_UPDOWN_AXIS, DIMOUSE_YAXIS, 0, "Move", },
    {INPUT_FIREWEAPONS, DIMOUSE_BUTTON0, 0, "Fire weapons", },
    {INPUT_ENABLESHIELD, DIMOUSE_BUTTON1, 0, "Enable shields", },
};
```

A DIACTIONFORMAT structure is then initialized to contain the **DIACTION** array *g_rgGameAction* defined above. The **DIACTIONFORMAT** structure also defines the genre, buffer size, axis scaling, and a friendly name for the action map. The *g_AppGuid* value is a GUID defined in the application to identify the action map. For more information on creating GUIDs, see Using GUIDS.

```
DIACTIONFORMAT diaf;
```

```
diaf.dwSize       = sizeof(DIACTIONFORMAT);
diaf.dwActionSize  = sizeof(DIACTION);
diaf.dwDataSize    = NUMBER_OF_ACTIONS * sizeof(DWORD);
diaf.dwNumActions  = NUMBER_OF_ACTIONS;
diaf.guidActionMap = g_AppGuid;
diaf.dwGenre       = DIVIRTUAL_SPACESIM;
diaf.rgoAction     = g_rgGameAction;
diaf.dwBufferSize  = 16;
diaf.lAxisMin      = -100;
diaf.lAxisMax      = 100;
diaf.tszActionMap  = "DI Test";
```

This **DIACTIONFORMAT** structure is matched against enumerated devices to find the best match in Step 3: Mapping Actions to Devices.

Step 3: Mapping Actions to Devices

Once the desired mapping has been defined, it must be applied to physical devices. IDirectInput8::EnumDevicesBySemantics begins this process in the following code fragment, where *m_pDI* is a previously defined IDirectInput8 object.

HRESULT hr;

```
hr = m_pDI->EnumDevicesBySemantics(m_strUserName, &m_diaf,
                        EnumSuitableDevicesCB, this, 0L);
```

The first parameter, *m_strUserName* is useful in the case of multiple users who might have different mapping preferences. However, in the case of this example, this value is NULL, indicating the user currently logged onto the system.

The second parameter is the address of the DIACTIONFORMAT structure containing the mapping information.

The third parameter is the address of a callback function that is called for each device found on the system. The function used in the sample is based on the **DIEnumDevicesBySemanticsCallback** function.

The fourth parameter could be any 32-bit value that you want to make available to the callback function. Here, this is a **this** pointer to the class object used by the MultiMapper sample for the various action-mapping setup and manipulation functions. This gives the callback function access to the class' functions.

The final parameter is used for flags. None are used in this example.

The **EnumSuitableDevicesCB** callback function calls the application-defined function **AddDevices**, which in turn calls the **IDirectInputDevice8::BuildActionMap** function on the device *pdidDevice* currently being enumerated. This call is shown in the following code fragment.

```
hr = pdidDevice->BuildActionMap( &m_diaf, m_strUserName, 0L );
```

The first parameter is once again the address of the **DIACTIONFORMAT** structure containing the action map.

The second parameter is again the name of the user if desired, though NULL in this case.

The final parameter holds control flags, with none being passed in this case.

BuildActionMap takes the list of control assignments specified in the **DIACTIONFORMAT** structure and attempts to map them to specific device objects on the device being enumerated. The results of the mapping are returned in the same structure. Each pass maps different controls depending on the nature of the currently enumerated device. A keyboard, for instance, maps controls that specify a mapping to a particular key, but leaves controls that specify buttons, sliders, or axes unmapped. If a joystick is enumerated next, **BuildActionMap** ignores those controls mapped to keyboard keys, but it attempts to map the buttons, sliders, and axes to the joystick's device objects.

The results of this default mapping can be examined and manipulated. This is done in Step 4: Configuring and Applying the Action Map.

Step 4: Configuring and Applying the Action Map

Once the mapping has been returned in the **DIACTIONFORMAT** structure, it can be manipulated before being applied by **IDirectInputDevice8::SetActionMap**.

The **dwHow** member of each **DIACTION** structure might be examined at this point to determine both whether the mapping was successful and what the criteria were in selecting the device object to which that action was mapped. For instance, you can determine if the mapping was specified by the hardware manufacturer or requested by the user.

The actual mapping itself may be changed as well by editing the **dwSemantic** member of the appropriate **DIACTION** structure returned in the default mapping. You can choose to make those changes, but it is not recommended that you do so.

Once the action map configuration is finalized, it must be applied to the device to bind the physical controls to the game actions. This is done by calling **SetActionMap** as follows:

```
hr = pdidDevice->SetActionMap( &m_diaf, m_strUserName, 0L );
```

Action maps, once created, can be displayed for and easily altered by the user. This is explained in Step 5: Displaying the Action Map.

Step 5: Displaying the Action Map

In addition to the advantage of action mapping in enabling the developer to define game actions independently of specific controller hardware, action mapping also benefits the user by providing a user interface (UI) for easy reconfiguration.

The UI can be displayed in two forms, text-only or graphics. In addition, each of those two options can be viewed in two modes: view-only or edit.

To display the UI, some preliminary setup is involved. First, a DICOLORSET structure is declared and initialized. This structure, as its name implies, describes the color set used to display the UI. The following code fragment shows the setup of this structure.

```
DICOLORSET dics;
ZeroMemory(&dics, sizeof(DICOLORSET));
dics.dwSize = sizeof(DICOLORSET);
```

By using the **ZeroMemory** function to set the memory block to zeroes, it indicates that the default color scheme should be used. Beyond this, the only element that requires initialization is the **dwSize** member, which is set equal to the size of the **DICOLORSET** structure.

The **DICOLORSET** structure is then included as part of a larger DICONFIGUREDEVICESPARAMS structure as seen in this code fragment.

```
DICONFIGUREDEVICESPARAMS dicdp;
ZeroMemory(&dicdp, sizeof(dicdp));
dicdp.dwSize = sizeof(dicdp);
dicdp.dwcUsers       = 1;
dicdp.lptszUserNames = m_strUserName;

dicdp.dwcFormats     = 1;
dicdp.lprgFormats    = &m_diaf;
dicdp.hwnd           = m_hWnd;
dicdp.lpUnkDDSTarget = NULL;
```

The **DICONFIGUREDEVICESPARAMS** structure is declared and its allotted memory is set to contain nothing but zeroes. Once again, the **dwSize** member is set to the size of the structure. Only the currently logged user is being used in this example, so **dwcUsers** is set to 1. Recall that the *m_strUserName* variable is set to NULL so that the user's name is the name of the user currently logged onto the system.

Only one DIACTIONFORMAT structure containing the action map is being passed, so **dwcFormats** is set to 1 and the address of the **DIACTIONFORMAT** structure is passed as **lprgFormats**.

The **hwnd** variable is assigned the handle of the application window and the **lpUnkDDSTarget** is set to NULL to indicate that the Microsoft® Windows®

Graphics Device Interface (GDI) functions should be used to draw the configuration user interface image.

The initialized **DICONFIGUREDEVICESPARAMS** structure is then passed to IDirectInput8::ConfigureDevices as illustrated in this code fragment.

```
hr = m_pDI->ConfigureDevices(NULL, &dicdp, dwFlags, NULL);
```

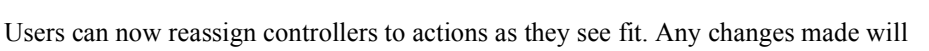
The first and last parameters, which are used with an optional callback function, are set to NULL as they are not used in this sample.

The second parameter is the address of the **DICONFIGUREDEVICESPARAMS** structure containing the data to be displayed and other auxiliary information.

The third parameter is used for flags that determine whether the display is in read-only or edit mode. If you want to display the configuration for informational purposes without giving the user the option of changing it, you use the DICD_DEFAULT flag in this parameter. However, in this sample the **dwFlags** variable had been set earlier to DICD_EDIT, so the display allows user configuration. This is discussed in Step 6: User Configuration of the Action Map.

Step 6: User Configuration of the Action Map

At this point, the user configuration interface is displayed. If the device manufacturer has provided a graphic, users see an image of the device with labels indicating the action assigned to each device object. If no graphic is available, a simple text list of available device objects with their associated actions is displayed. An example of the former is shown here.



Users can now reassign controllers to actions as they see fit. Any changes made will be reflected in the DIACTIONFORMAT structure. Once the user interface is closed, the new action map must be sent to IDirectInputDevice8::SetActionMap before it can be implemented.

Using the data associated with the controls in an action map is described in Step 7: Retrieval of Action Mapped Data.

Step 7: Retrieval of Action Mapped Data

Action mapping was designed with buffered data in mind and will most often be used in that manner. The only difference between retrieving data from an action-mapped device and a device with no mapping is where you get it. Without action mapping, you would look to the **dwOfs** member of the DIDEVICEOBJECTDATA structure to determine which device object reported a change. However, with action mapping, you look to the **uAppData** member. Data is found in the **dwData** member.

For more detailed information on the retrieval of action-mapped data, see Retrieving Action Data.

DirectInput Visual Basic Tutorials

This section contains the following tutorials, each providing step-by-step instructions for implementing Microsoft® DirectInput® in a Microsoft Visual Basic® application.

- Tutorial 1: Using the Keyboard

  The first tutorial shows how to add DirectInput keyboard support to an existing application, using event polling.
- Tutorial 2: Using the Mouse

  The next tutorial takes you through the steps of providing DirectInput mouse support in an application, using the exclusive cooperative level and event notification. The tutorial is based on the ScrawlB sample, and focuses on buffered data.
- Tutorial 3: Using the Joystick

  This tutorial shows how to enumerate all the joysticks connected to a system, how to create and initialize DirectInputDevice objects for each of them in a callback function, and how to retrieve immediate data.
- Tutorial 4: Using Force Feedback

  This tutorial shows how to implement simple force-feedback effects in an application.
- Tutorial 5: Using Action Mapping

  The final tutorial shows how to set up, apply, and use the new Action Mapping functionality of DirectInput.

Tutorial 1: Using the Keyboard

To prepare for keyboard input, you first create a **DirectInput8** object. Then you create a **DirectInputDevice8** object representing the keyboard. The **DirectInputDevice8** class methods are used to set the behavior of the device and retrieve data.

The tutorial divides the required tasks into the following steps.

- Step 1: Creating the DirectInput8 Object and the Keyboard Device
- Step 2: Setting the Keyboard Parameters
- Step 3: Gaining Access to the Keyboard
- Step 4: Retrieving Immediate Data from the Keyboard
- Step 5: Closing Down the DirectInput System

Step 1: Creating the DirectInput8 Object and the Keyboard Device

The first step in setting up the Microsoft® DirectInput® system is to create a single **DirectInput8** object as overall manager. This is done with a call to the **DirectX8.DirectInputCreate** method, typically in the **Load** event handler for the main form or in **Sub Main**.

```
Dim dx As New DirectX8
Dim di As DirectInput8

Set di = dx.DirectInputCreate()
```

The keyboard is then created as a standard device by passing the keyboard GUID alias to **DirectInput8.CreateDevice**.

```
Dim didev As DirectInputDevice8
Set didev = di.CreateDevice("GUID_SysKeyboard")
```

Once the keyboard device is created, its methods will be used to set its data format and cooperative level in Step 2: Setting the Keyboard Parameters.

Step 2: Setting the Keyboard Parameters

After creating a **DirectInputDevice8**, your application must set the device's data format. For keyboards, as with other standard devices, this is a very simple task. Call the **DirectInputDevice8.SetCommonDataFormat** method, specifying the data format provided by Microsoft® DirectInput® by using the DIFORMAT_KEYBOARD constant as the parameter. You must set the data format even if you intend to retrieve buffered data.

```
Call didev.SetCommonDataFormat(DIFORMAT_KEYBOARD)
```

DirectInput identifies the device objects by their offset within the data format. In the case of the keyboard, keys are identified by their offsets within the **DIKEYBOARDSTATE** type.

Before your application can gain access to the keyboard, it must set the device's behavior using the **DirectInputDevice8.SetCooperativeLevel** method, as follows:

```
didev.SetCooperativeLevel Me.hWnd, _
    DISCL_NONEXCLUSIVE Or DISCL_BACKGROUND
```

This method accepts the handle to the window to be associated with the device, and exactly two flags (either DISCL_EXCLUSIVE or DISCL_NONEXCLUSIVE, plus either DISCL_FOREGROUND or DISCL_BACKGROUND), indicating the desired cooperative level. However, DirectInput does not support exclusive access to

keyboard devices, so the DISCL_NONEXCLUSIVE flag must always be used in the case of keyboards as it is here.

The example also sets the background cooperative level, so input will be available regardless of whether the form is in the foreground. Note also that keystrokes continue to be passed through to whatever application has the focus. Most applications don't need input when they're in the background, and in such cases the DISCL_FOREGROUND flag should be set instead.

Now that the keyboard device has been created and set up, it must be acquired. This is shown in Step 3: Gaining Access to the Keyboard.

Step 3: Gaining Access to the Keyboard

After your application sets the keyboard's behavior, it can acquire access to the device by calling the **DirectInputDevice8.Acquire** method. The application must acquire the device before retrieving data from it. The **Acquire** method accepts no parameters.

```
Call didev.Acquire
```

In the sample, the application is unlikely to fail to acquire the keyboard, or to lose it later, because it is using the background, nonexclusive cooperative level. However, in general it is good practice to handle errors on calls to **Acquire** as well as when attempting to retrieve data.

You are now ready to begin receiving data from the keyboard. This is shown in Step 4: Retrieving Immediate Data from the Keyboard.

Step 4: Retrieving Immediate Data from the Keyboard

Once a device is acquired, your application can start retrieving data from it. The simplest way to do this is to call the **DirectInputDevice8.GetDeviceStateKeyboard** method, which takes a snapshot of the device's state at the time of the call.

The **GetDeviceStateKeyboard** method accepts as its single parameter a **DIKEYBOARDSTATE** type, which simply contains an array of 256 bytes.

The following sample code retrieves the state of the keyboard.

```
Dim state As DIKEYBOARDSTATE
Call didev.GetDeviceStateKeyboard(state)
```

After retrieving the keyboard's current state, your application may respond to specific keys that were pressed at the time of the call. Each element in the buffer represents a key. If an element's high bit is set equal to 1, the key was pressed at the moment of the call. Otherwise, the key was up. To check the state of a given key, use the constants of the **CONST_DIKEYFLAGS** enumeration to index the buffer for a given key.

The following sample code shows how an application might move a vehicle around in response to the arrow keys.

```
If state.Key(DIK_UP) And &H80 Then
  ' Move the vehicle up
End If
if state.Key(DIK_DOWN) And &H80 Then
  ' Move the vehicle down
End If
' And so on.
```

The following form could also be used.

```
If state.Key(DIK_UP) Then
  ' Move the vehicle up
End If
```

Keep in mind that DIK_UP is a single key, the dedicated up arrow key. DirectInput treats the 8 key on the numerical keypad as a distinct key, and gives it the same identifier regardless of whether NUM LOCK is on. In order to allow input from either of the arrow keys, you would have to write code as in the following sample.

```
If state.Key(DIK_UP) Or state.Key(DIK_NUMPAD8) Then
  ' Move the vehicle up
End If
```

Step 5: Closing Down the DirectInput System

When an application is about to close, it should release all Microsoft® DirectInput® devices. This is done using the **DirectInputDevice8.Unacquire** method, generally in the **Unload** event handler as shown in the following code sample.

```
Private Sub Form_Unload(Cancel As Integer)
   didev.Unacquire
End Sub
```

Tutorial 2: Using the Mouse

This tutorial focuses on using the mouse at the exclusive cooperative level and shows how to retrieve buffered data in response to notifications. The sample code is based on the ScrawlB sample.

The tutorial is divided into the following steps.

- Step 1: Setting Up the Mouse
- Step 2: Setting Up Notifications
- Step 3: Managing Exclusive Access to the Mouse

- Step 4: Retrieving Buffered Data from the Mouse

Step 1: Setting Up the Mouse

First steps in setting up the mouse for use under Microsoft® DirectInput® are similar to those taken in Tutorial 1: Using the Keyboard. In the ScrawlB sample, initialization takes place in **Sub Main** after some global declarations.

```
Public objDX As New DirectX8
Public objDI As DirectInput8
.
.
Set objDIDev = objDI.CreateDevice("guid_SysMouse")
Call objDIDev.SetCommonDataFormat(DIFORMAT_MOUSE)
Call objDIDev.SetCooperativeLevel(frmCanvas.hwnd, _
      DISCL_FOREGROUND Or DISCL_EXCLUSIVE)
```

As opposed to the instance of the keyboard, DirectInput takes exclusive control of the mouse. The result is that as long as the application has the mouse in the acquired state, Microsoft Windows® does not generate mouse messages or display the system cursor. Note that DISCL_EXCLUSIVE must be combined with DISCL_FOREGROUND; it is not possible for an application to have exclusive access to the mouse and also receive input when it loses the focus.

Because the ScrawlB sample application is taking full responsibility for the mouse, it must also track the position of its private cursor, and it must scale movement of the cursor to movements of the mouse. The following global variables are used to store the cursor coordinates (in pixels relative to the upper left corner of the main form) and movement scaling.

```
Public g_cursorx As Long
Public g_cursory As Long
Public g_Sensitivity
```

In **Sub Main**, the application sets the buffer size so that it can receive buffered data, using the **DirectInputDevice8.SetProperty** method.

```
Dim diProp As DIPROPLONG
diProp.lHow = DIPH_DEVICE
diProp.lObj = 0
diProp.lData = BufferSize    ' BufferSize is a constant

Call objDIDev.SetProperty("DIPROP_BUFFERSIZE", diProp)
```

You are now ready to specify how mouse events will trigger data retrieval in Step 2: Setting Up Notifications

Step 2: Setting Up Notifications

Rather than polling for mouse input in **Sub Main**, the ScrawlB sample application relies on Microsoft® DirectInput® to notify it whenever a mouse event takes place. As part of initialization, the application gets an event handle and passes it to **DirectInputDevice8.SetEventNotification**.

```
Dim EventHandle As Long

EventHandle = objDX.CreateEvent(frmCanvas)
Call objDIDev.SetEventNotification(EventHandle)
```

DirectInput will now notify any object that implements the **DirectXEvent8** class. In the case of ScrawlB, the implementing object is the *frmCanvas* form, whose declarations section contains the following line.

```
Implements DirectXEvent8
```

This line causes *frmCanvas* to inherit all the methods of the **DirectXEvent8** class. The only method implemented by that class is **DirectXEvent8.DXCallback**, and the inheriting class must implement this method. DirectInput calls this method whenever an input event is signaled.

The implementation of **DXCallback** is covered under Step 4: Retrieving Buffered Data from the Mouse.

There are responsibilities involved in maintaining an exclusive access level. They are covered in Step 3: Managing Exclusive Access to the Mouse

Step 3: Managing Exclusive Access to the Mouse

The ScrawlB sample demonstrates using the mouse with the exclusive foreground cooperative level. At this level, Microsoft® Windows® does not track the mouse while it is acquired by the application. There are three major consequences for the application.

- It is entirely responsible for displaying a cursor if one is needed, and for moving it at an appropriate speed in response to Microsoft DirectInput® data.
- It loses acquisition when the user switches to another application by using the keyboard, and must reacquire the mouse when the user switches back.
- It must provide a means for Windows to get the mouse back whenever the user needs to use the system cursor—for example, to navigate a menu within the application.

ScrawlB handles the first responsibility by keeping a private record of mouse movements, adjusted by a user-defined sensitivity value, and displaying an icon that serves as a cursor for drawing.

ScrawlB loses acquisition of the mouse involuntarily whenever the application window moves to the background. When it comes to the foreground again, the application automatically reacquires the mouse by calling **DirectInputDevice8.Acquire** in the MouseMove event handler, which is called whenever Windows sends a mouse message to the application. This happens as soon as the system cursor moves over the client window, or the client window gains the focus under the system cursor. Once ScrawlB reacquires the mouse, of course, no more Windows mouse messages are sent, so the **Form_MouseMove** method is not called in response to subsequent mouse events.

In order to enable the user to navigate the shortcut menu, or to display the system cursor for some other purpose such as resizing the window, ScrawlB calls the **DirectInputDevice8.Unacquire** method whenever the user opens the menu. When the menu is closed, **Form_MouseMove** is called and the mouse is reacquired, unless the user has chosen **Suspend** from the menu, in which case a flag is set. This flag prevents **Form_MouseMove** from reacquiring, so that the user can continue using the system cursor.

The sample application demonstrates one more technique for releasing the mouse. If the user opens the system menu by pressing ALT+SPACE, Windows sends a WM_ENTERMENULOOP message. This message is intercepted by a subclassed window procedure, which then unacquires the mouse. As long as Windows is using the system cursor for navigating the menu or enabling the user to move or resize the window, it sends no mouse messages to *frmCanvas*, so the application doesn't attempt to reacquire the mouse in **Form_MouseMove**. A similar technique could be used for intercepting other Windows messages such as WM_ACTIVATE or WM_ACTIVATEAPP, so that the application could fully control acquisition and unacquisition of the mouse in response to gaining and losing the focus.

You are now ready to receive data from the mouse in Step 4: Retrieving Buffered Data from the Mouse

Step 4: Retrieving Buffered Data from the Mouse

The ScrawlB sample retrieves buffered mouse data inside the **DirectXEvent8.DXCallback** method implemented by *frmCanvas*. This method is called each time an input event is signaled.

The method declares an array of the same size as the buffer Microsoft® DirectInput® is using to store the data. This is the size that was set previously by a call to **DirectInputDevice8.SetProperty**.

```
Dim diDeviceData(1 To BufferSize) As DIDEVICEOBJECTDATA
```

Also required are a variable to receive the number of items actually retrieved, a loop counter, and a variable to track the previous sequence number of an event.

```
Dim NumItems As Integer
```

```
Dim i As Integer
Static OldSequence As Long
```

The application now retrieves all the data available, in a single call to **DirectInputDevice8.GetDeviceData**.

```
On Error GoTo INPUTLOST
NumItems = objDIDev.GetDeviceData(diDeviceData, 0)
On Error GoTo 0
```

Note the error trap. One of the events that DirectInput will signal is loss of acquisition. If the user switches to another application, for instance, ScrawlB will no longer have the mouse in the acquired state. An event will be signaled and **GetDeviceData** will be called, but the method will fail because data can be retrieved only from a device that is in an acquired state.

The application next iterates through the retrieved items and examines the data in the **DIDEVICEOBJECTDATA** type, comparing the **lOfs** member against the constants for the various buttons and axes that are of interest. For the x-axis, for instance, the application extracts the change in axis position from the **lData** member and uses this to adjust the cursor position, taking into account the user-defined sensitivity, as in the following code sample.

```
For i = 1 To NumItems
   Select Case diDeviceData(i).lOfs
      Case DIMOFS_X
         g_cursorx = g_cursorx + diDeviceData(i).lData * _
            g_Sensitivity
```

The application also examines the sequence number and compares it with the previous one. If two axis events have the same sequence number, the mouse has been moved diagonally. To avoid a staircase effect, it is not desirable to update the cursor position or draw a line until both movements have been taken into account.

```
      If OldSequence <> diDeviceData(i).lSequence Then
         UpdateCursor
         OldSequence = diDeviceData(i).lSequence
      Else
         OldSequence = 0
      End If
```

For the buttons, the method determines the event type by checking the appropriate bit in **lData**. If the bit is set, the mouse button was pressed; otherwise, it was released. Remember, **GetDeviceData** does not return the current state of the device, so it is up to the application to keep a private record of whether a button is being held down. For the left button, ScrawlB keeps this information in the *Drawing* variable.

```
      Case DIMOFS_BUTTON0
         If diDeviceData(i).lData And &H80 Then
```

```
            Drawing = True

            ' Keep record for Line function
            CurrentX = g_cursorx
            CurrentY = g_cursory

            ' Draw point in case button-up follows immediately
            PSet (g_cursorx, g_cursory)
          Else
            Drawing = False
          End If
.
.
    End Select
Next i
```

Tutorial 3: Using the Joystick

This tutorial shows how to create a joystick device, set its properties, and retrieve immediate data. The example code is based on the Joystick sample.

The tutorial divides the required tasks into the following steps.

- Step 1: Enumerating Devices and Creating the Joystick
- Step 2: Getting Joystick Capabilities
- Step 3: Setting Joystick Properties
- Step 4: Retrieving Immediate Data from the Joystick

Step 1: Enumerating Devices and Creating the Joystick

Because there is no such thing as a system joystick in the same sense that there can be a system keyboard or mouse, in order to create a **DirectInputDevice8** object for a joystick you first need to obtain an instance GUID, or globally unique identifier. Generally, this is done by enumerating the available joysticks, presenting the user with a choice, and then obtaining the information for the selected device.

As with all devices, the DirectInput system must first be created.

```
Dim dx As New DirectX8
Dim di As DirectInput8

Set di = dx.DirectInputCreate()
```

The next step is enumeration of devices using the **DirectInput8.GetDIDevices** method. As a parameter to this method, flags are passed indicating that the application

should enumerate only game controller devices that are physically attached to the system. The results of this enumeration are stored in an instance of the **DirectInputEnumDevices8** class.

```
Dim diDev As DirectInputDevice8
Dim diDevEnum As DirectInputEnumDevices8

Set diDevEnum = di.GetDIDevices( _
        DI8DEVCLASS_GAMECTRL, DIEDFL_ATTACHEDONLY)
```

Once the enumeration is complete, assuming at least one joystick was found, that information can be manipulated. In the case of the Joystick sample, the devices are added to a list box from which the user can make a choice. The **DirectInputEnumDevices8.GetCount** method specifies how many items are contained in the enumeration.

```
'Add attached joysticks to the list box
Dim i As Integer
For i = 1 To diDevEnum.GetCount
   Call lstJoySticks.AddItem( _
      diDevEnum.GetItem(i).GetInstanceName)
Next
```

GetInstance returns a friendly name for the device.

At this point, the Joystick sample initializes an event handle to associate with the device. This is used at the end of Step 2: Getting Joystick Capabilities.

```
Dim EventHandle as Long
EventHandle = dx.CreateEvent(Me)
```

When the user selects a joystick from the list, the device is created and initialized in the lstJoySticks_Click procedure. Because there are no standard GUIDs to pass to **DirectInput8.CreateDevice**, as there are for the system keyboard and system mouse, the GUID of the device instance must be extracted by using the **DirectInputDeviceInstance8.GetGuidInstance** method. Note that as with all enumerated collections in Microsoft DirectX® for Microsoft Visual Basic®, the device enumeration is 1-based. Therefore, the index is one greater than the index of the selected item in the list box. The list box must also be unsorted.

```
Set diDev = di.CreateDevice(diDevEnum.GetItem( _
      lstJoySticks.ListIndex + 1).GetGuidInstance)
```

The joystick must also be initialized for cooperative level and format. In this case, the DIFORMAT_JOYSTICK format is used. Another format, DIFORMAT_JOYSTICK2 is also available for joysticks with a larger assortment of controls.

```
diDev.SetCommonDataFormat DIFORMAT_JOYSTICK
diDev.SetCooperativeLevel Me.hWnd, _
```

```
DISCL_BACKGROUND Or DISCL_NONEXCLUSIVE
```

You must set the data format before attempting to enumerate objects on the device or manipulate its properties.

Once the joystick is in place, its capabilities are explored in Step 2: Getting Joystick Capabilities

Step 2: Getting Joystick Capabilities

Getting basic information about the buttons, axes, and point-of-view controllers on the device requires a simple call to **DirectInputDevice8.GetCapabilities**.

```
Dim joyCaps As DIDEVCAPS
diDev.GetCapabilities joyCaps
```

The Joystick sample needs only the number of buttons and point-of-view controllers on the device. Although the **DIDEVCAPS** type also reports the number of axes on the device, it does not reveal anything about what those axes are. For this information, the sample calls its own IdentifyAxes procedure. It first declares its variables and initializes an array that will hold Boolean values indicating the presence or absence of each possible axis.

```
Sub IdentifyAxes(diDev As DirectInputDevice8)

    Dim didoEnum As DirectInputEnumDeviceObjects
    Dim dido As DirectInputDeviceObjectInstance
    Dim i As Integer

    For i = 1 To 8
       AxisPresent(i) = False
    Next
```

The procedure goes on to enumerate device objects on the device. The DIDFT_AXIS value will restrict the enumeration to axes.

```
    Set didoEnum = diDev.GetDeviceObjectsEnum(DIDFT_AXIS)
```

Each **DirectInputDeviceObjectInstance** is then queried for its offset within the data format that was established earlier by the call to **DirectInputDevice8.SetCommonDataFormat**. This offset identifies the conventional role or type of the axis; for instance, a GUID_RzAxis likely corresponds to a twisting motion on the main stick. Keep in mind, though, that device drivers are free to assign any designation to an axis. It is always a good idea to allow users to change the mapping of the axes to actions within your application.

```
    Dim sGuid as String
    For i = 1 To didoEnum.GetCount
```

```
        Set dido = didoEnum.GetItem(i)
        sGuid = didoEnum.GetItem(i)

        Select Case sGuid
            Case "GUID_Xaxis"
                AxisPresent(1) = True
            Case "GUID_Yaxis"
                AxisPresent(2) = True
' and so on
.
.
.
        End Select
    Next
End Sub
```

The application now passes the event handle created earlier to the device, so that notifications will be sent to the form when an input event takes place.

```
Call diDev.SetEventNotification(EventHandle)
```

Next, you set the joystick properties in Step 3: Setting Joystick Properties.

Step 3: Setting Joystick Properties

Property changes are made through the **DirectInputDevice8.SetProperty** method, and must be made after the data format of the device is established, but before it is acquired.

Because different devices might return different ranges of axis values, it is a good idea to set a range that will apply to all devices. The Joystick sample requests that all axis values be within the range 0 to 10,000.

Properties are set using different types for different properties. For an axis value, use the **DIPROPRANGE** type. This property type structure is then applied by calling **SetProperty**.

```
Dim DiProp_Range As DIPROPRANGE

With DiProp_Range
    .lHow = DIPH_DEVICE
    .lMin = 0
    .lMax = 10000
End With
diDev.SetProperty "DIPROP_RANGE", DiProp_Range
```

The DIPH_DEVICE flag is used to indicate that the property applies to all axes on the device. The Joystick sample also sets the dead zone and saturation zones for the

joystick. In this case, it is not desirable to set these values for other axes such as a throttle or rudder. The **lHow** member of the property type is set to indicate that the property change applies to a device object identified by its offset within the data format, and the **lObj** member is set to that offset value. In this case, the DIPROPLONG type is used to send values of type Long through **SetProperty**. The following sample code demonstrates this for the dead zone property.

```
Dim DiProp_Dead As DIPROPLONG

With DiProp_Dead
   .lData = 1000
   .lHow = DIPH_BYOFFSET

' Set for x-axis
   .lObj = DIJOFS_X
   diDev.SetProperty "DIPROP_DEADZONE", DiProp_Dead

' Set for y-axis
   .lObj = DIJOFS_Y
   diDev.SetProperty "DIPROP_DEADZONE", DiProp_Dead
End With
```

The value in **lData** is the proportion of the range of travel, in units of 10,000, that is set up as a dead zone. Because 1000 is one-tenth of 10,000, it specifies that one-tenth of the range of travel of the x-axis and y-axis, balanced at the center, will be reported as still being in the center (in this case, a value of 5000).

Note

Dead zone and saturation values are always calculated in units of 10,000, regardless of the actual range of values reported by the device.

The saturation zones for the device are set similarly. In this case, **lData** is 9500, because it represents the proportion of the range of travel that lies outside the saturation zones at the extremities. In other words, the top and bottom 5 percent of the raw values returned by the joystick is reported as the minimum or maximum range value set for the device (in this case, 0 or 10,000).

You are now ready to get data from the device in Step 4: Retrieving Immediate Data from the Joystick.

Step 4: Retrieving Immediate Data from the Joystick

Now that the necessary properties have been set, the device can be acquired and data can be collected from the joystick. In the Joystick sample, the device is polled in a loop after its properties have been set. This uses the **DirectInputDevice8.Poll**

method. In a real-world multimedia application, polling would take place in the main game loop or rendering loop.

```
diDev.Acquire

While DoEvents
   diDev.Poll
Wend
```

The sample application does not actually retrieve immediate data each time the device is polled. To avoid unnecessary screen updates, it relies on notification, and the call to **Poll** is only to ensure that notifications are issued by devices that do not generate interrupts. Polling is not necessary for some game devices (such as HIDs), but it is as efficient to make a redundant call to **DirectInputDevice8.Poll** as to check a flag for the device before calling **Poll**.

Notifications are handled, as usual, in the implementation of the **DirectXEvent8.DXCallback** method. Each time this procedure is called, the application knows that a joystick event has occurred. At this point the application could call **DirectInputDevice8.GetDeviceData** to retrieve any pending axis changes or button events. However, with a game controller it is more common to retrieve the state of the entire device. Therefore, the sample uses **DirectInputDevice8.GetDeviceStateJoystick**, called from within the callback function.

```
Dim js As DIJOYSTATE

diDev.GetDeviceStateJoystick js
```

The values in the **DIJOYSTATE** type contain the state of all the device objects on the joystick. Because Step 2: Getting Joystick Capabilities checked which buttons, axes, and point-of-view controllers are actually present, you know which members in the type to ignore.

Tutorial 4: Using Force Feedback

This tutorial shows how to test for the presence of a force-feedback device, how to create and play an effect, and how to change the parameters of an effect as it is playing.

The tutorial divides the tasks into the following steps.

- Step 1: Initializing the Force-Feedback Device
- Step 2: Setting Device Properties
- Step 3: Creating an Effect
- Step 4: Modifying an Effect

Step 1: Initializing the Force-Feedback Device

In order to implement force feedback, you must first determine whether an appropriate device is available, and if so, create a **DirectInputDevice8** object for it. This tutorial presumes an application requiring a device that can play effects on both the x-axis and y-axis.

The following example starts, as usual, by creating a **DirectInput8** object, then using that object to enumerate the available devices into a **DirectInputEnumDevices8** object. The enumeration is restricted to attached devices of type DI8DEVTYPE_JOYSTICK.

```
Dim dx As DirectX8
Dim di As DirectInput8

Dim diDev As DirectInputDevice8
Dim diEnumDev As DirectInputEnumDevices8
Dim diDevInst As DirectInputDeviceInstance8
Dim diDevObjEnum As DirectInputEnumDeviceObjects
Dim devobj As DirectInputDeviceObjectInstance

Dim devcaps As DIDEVCAPS
Dim ForceX As Boolean, ForceY As Boolean
Dim FoundForce As Boolean
Dim efftype As Long
Dim strObjGuid As String

Dim i As Integer, iAxes As Integer

Set di = dx.DirectInputCreate
Set diEnumDev = di.GetDIDevices(DI8DEVTYPE_JOYSTICK, _
                                DIEDFL_ATTACHEDONLY)
```

The example now iterates through the available devices until it finds one that has force-feedback capabilities.

```
For i = 1 To diEnumDev.GetCount
   Set diDevInst = diEnumDev.GetItem(i)
   Set diDev = di.CreateDevice(diDevInst.GetGuidInstance)
   Call diDev.GetCapabilities(devcaps)
   If devcaps.lFlags And DIDC_FORCEFEEDBACK Then
   'Process the force-feedback device
```

Next, because the example requires that both the X and Y axes must support force feedback, all axes on the device are enumerated in a **DirectInputEnumDeviceObjects** collection.

```
   Set diDevObjEnum = didev.GetDeviceObjectsEnum(DIDFT_AXIS)
```

The collection of axes present is then inspected, looking for the x-axis and y-axis, and checking to see whether they support force-feedback effects.

```
    ForceX = False
    ForceY = False
    For iAxes = 1 To diDevObjEnum.GetCount
       Set devobj = diDevObjEnum.GetItem(iAxes)
       strObjGuid = devobj.GetGuidType
       If strObjGuid = "GUID_XAxis" Then
          If devobj.GetFlags And DIDOI_FFACTUATOR Then
             ForceX = True
          End If
       ElseIf strObjGuid = "GUID_YAxis" Then
          If devobj.GetFlags And DIDOI_FFACTUATOR Then
             ForceY = True
          End If
       End If
       FoundForce = ForceX And ForceY
       If FoundForce Then Exit For
    Next iAxes

  Next i    'Next device

  If Not FoundForce Then
     MsgBox "Two force feedback axes required."
  End If
```

Now that a suitable device is assured, its properties are set in Step 2: Setting Device Properties.

Step 2: Setting Device Properties

Presuming that you have found a suitable force-feedback device, the **DirectInputDevice8** object created in the last step in order to check capabilities can be left in place. It requires some further normal initialization, as covered in Tutorial 3: Using the Joystick: setting the data format and cooperative level, and setting the range properties for input. Note that the cooperative level must include DISCL_EXCLUSIVE. Although an actual application probably does not require background access to the device, the DISCL_BACKGROUND flag makes debugging easier because you will not lose acquisition when you switch to the code window.

```
  Call didev.SetCommonDataFormat(DIFORMAT_JOYSTICK2)
  Call didev.SetCooperativeLevel(Me.hWnd, _
       DISCL_BACKGROUND Or DISCL_EXCLUSIVE)
  ' Set range properties...
```

For force feedback, it is often good practice to disable the autocenter property of the device, so that the default spring action will not interfere with effects you create. This is done by sending a **DIPROPLONG** type through the DirectInputDevice8.SetProperty method, setting the DIPROP_AUTOCENTER property to 0.

```
Dim prop As DIPROPLONG
prop.lData = 0
prop.lHow = DIPH_DEVICE
prop.lObj = 0
Call didev.SetProperty("DIPROP_AUTOCENTER", prop)
```

Once you have configured the device as desired, it can be acquired.

```
didev.Acquire
```

Now that you have a force-feedback device ready, you need to create an effect to use with it. This is covered in Step 3: Creating an Effect.

Step 3: Creating an Effect

The effect created in this example simulates the pull of a stationary object, such as a magnet, on a movable object, such as an iron ball. The movable object is controlled by the joystick. At the beginning of the simulation, the iron ball is due south of the magnet and experiencing one-half the maximum pull.

The pull of the magnet is simulated by a constant force. The first step is to set up the **DIEFFECT** type that describes the force.

```
Dim EffectInfo as DIEFFECT
With EffectInfo
   .constantForce.lMagnitude = 5000
   .lDuration = -1      ' Infinite
   .x = 18000
   .lGain = 10000       ' Play at full magnitude
   .lTriggerButton = -1  ' No trigger button
End With
```

By setting the **x** parameter to 18,000, you specify a force originating from a spot 180 degrees from due north. In other words, a force moving from due south toward due north, away from the user. For an in-depth discussion of effect directions, see Effect Direction

All other members of **DIEFFECT** are either not relevant to a constant force or are valid with a value of 0.

Now that the effect has been defined, it can be created using the **DirectInputDevice8.CreateEffect** method.

```
Dim di_effect As DirectInputEffect
```

```
Set di_effect = didev.CreateEffect("GUID_ConstantForce", EffectInfo)
```

If the device is in an acquired state at this time, the effect is automatically downloaded. If not, the effect will be downloaded when it is started using the **DirectInputEffect.Start** method.

```
di_effect.Start(1, 0)
```

Only one iteration of the effect is needed because it has infinite duration and no flags are required.

While a constant unchanging effect can be useful, an effect might also change as it plays or in reaction to user input. This is discussed in Step 4: Modifying an Effect.

Step 4: Modifying an Effect

As the user moves the iron ball around using the joystick, the application adjusts the direction and magnitude of the effect. First, however, you should check whether the device enables changing the direction of the effect without stopping and restarting the effect. Microsoft® DirectInput® will stop and restart the effect automatically if necessary, but by doing this yourself, you might avoid unseemly breaks in continuity.

In order to examine the capabilities for any effect, you first enumerate supported effects. To do so, use the **DirectInputDevice8.GetEffectsEnum** method, which resolves to a **DirectInputEnumEffects** type. The DIEFT_CONSTANTFORCE flag enables you to restrict the enumeration to supported only constant force effects.

```
Dim diEnumEffects As DirectInputEnumEffects
Set diEnumEffects = didev.GetEffectsEnum(DIEFT_CONSTANTFORCE)
```

Next, examine the enumeration for the particular effect you are interested in. If the enumerated effect meets the criteria, call the **DirectInputEnumEffects.GetDynamicParams** method to examine the effect for parameters that can be changed without stopping and restarting. Examine the flags returned by **GetDynamicParams**, comparing them against known values from CONST_DIEPFLAGS. In the case of the following example, the *params* variable containing the result flags is compared against DIEP_DIRECTION to ensure that the direction of the effect can be changed while the effect is running.

```
Dim params As Long
Dim i As Long

For i = 1 To diEnumEffects.GetCount
   ' Look for the standard constant force. There could be others.

   If diEnumEffects.GetEffectGuid(i) = "GUID_ConstantForce" Then
      params = diEnumEffects.GetDynamicParams(i)
      Exit For
   End If
```

```
Next i

If Not (params And DIEP_DIRECTION) Then
    ' Cannot change direction dynamically. Take remedial action such
    ' as limiting points at which modifications will be made.
End If
```

Similarly, to learn whether you can change the magnitude of the effect dynamically, perform the following check.

```
If Not (params And DIEP_TYPESPECIFICPARAMS)
   ' Cannot change type-specific parameters dynamically.
End If
```

The application modifies the effect each time the device has been polled for input and the iron ball has moved. Assume that the new magnitude is stored in *CurrentMag*, and the new direction (in degrees) in *CurrentBearing*. To make the necessary changes, you need to initialize only those members of **DIEFFECT** that are relevant.

```
EffectInfo.constantForce.lMagnitude = CurrentMag
EffectInfo.x = CurrentBearing * DI_DEGREES
```

You now pass the **DIEFFECT** type with the new data to **DirectInputEffect.SetParameters** along with flags indicating which members contain valid data.

```
di_effect.SetParameters(EffectInfo, _
      DIEP_DIRECTION Or DIEP_TYPESPECIFICPARAMS)
```

The DIEP_TYPESPECIFICPARAMS flag indicates that any member containing information particular to that type of effect is valid. In the case of a constant force, this means the **constantForce** member.

Tutorial 5: Using Action Mapping

This tutorial guides you through the setup, modification, and application of an action map. Code samples are based on the ActionMapper sample, which is designed as though it were a space combat game.

Preliminary steps of setting up the Microsoft® DirectInput® system are the same for any application and are covered in detail in the other tutorials. The keyboard, mouse, and a joystick (if present) are all used in the sample, so each of those devices needs to be set up.

The tutorial divides the tasks into the following steps.

- Step 1: Defining the Game Actions
- Step 2: Defining the Action Map

- Step 3: Mapping Actions to Devices
- Step 4: Configuring and Applying the Action Map
- Step 5: Displaying the Action Map
- Step 6: User Configuration of the Action Map
- Step 7: Retrieval of Action Mapped Data

Step 1: Defining the Game Actions

The first step toward creating an action map is the definition of the game actions. These actions are specific to the application and are defined by the developer. In the case of the ActionMapper sample, the actions chosen reflect its needs as a space combat simulator. These values can be defined in many ways as the developer sees fit. They could be defined as an enumeration, as in the Preparing the Action Map overview, or they could be defined as constants, as in the following code fragment from the ActionMapper sample.

```
Const INPUT_LEFTRIGHT_AXIS = 1
Const INPUT_UPDOWN_AXIS = 2
Const INPUT_TURNLEFT = 4
Const INPUT_TURNRIGHT = 5
Const INPUT_FORWARDTHRUST = 6
Const INPUT_REVERSETHRUST = 7
Const INPUT_FIREWEAPONS = 8
Const INPUT_ENABLESHIELD = 9
Const INPUT_DISPLAYGAMEMENU = 10
Const INPUT_QUITGAME = 11
```

Once these values are defined, they can be assigned to genre-specific action-mapping constants. This is shown in Step 2: Defining the Action Map.

Step 2: Defining the Action Map

Next, each game action should be assigned to at least one virtual control or device object. These controls are defined as action-mapping constants, and they are defined by genre and subgenre. In the case of the ActionMapper sample, the best choice is the Space Combat subgenre of the flight games genre. This subgenre makes the following constants available.

Priority 1 Controls

DIAXIS_SPACESIM_LATERAL
DIAXIS_SPACESIM_MOVE
DIAXIS_SPACESIM_THROTTLE
DIBUTTON_SPACESIM_FIRE
DIBUTTON_SPACESIM_MENU

DIBUTTON_SPACESIM_TARGET
DIBUTTON_SPACESIM_WEAPONS

Priority 2 Controls

DIAXIS_SPACESIM_CLIMB
DIAXIS_SPACESIM_ROTATE
DIBUTTON_SPACESIM_BACKWARD_LINK
DIBUTTON_SPACESIM_DEVICE DIBUTTON_SPACESIM_DISPLAY
DIBUTTON_SPACESIM_FASTER_LINK
DIBUTTON_SPACESIM_FIRESECONDARY
DIBUTTON_SPACESIM_FORWARD_LINK
DIBUTTON_SPACESIM_GEAR
DIBUTTON_SPACESIM_GLANCE_DOWN_LINK
DIBUTTON_SPACESIM_GLANCE_LEFT_LINK
DIBUTTON_SPACESIM_GLANCE_RIGHT_LINK
DIBUTTON_SPACESIM_GLANCE_UP_LINK
DIBUTTON_SPACESIM_LEFT_LINK DIBUTTON_SPACESIM_LOWER
DIBUTTON_SPACESIM_PAUSE DIBUTTON_SPACESIM_RAISE
DIBUTTON_SPACESIM_RIGHT_LINK
DIBUTTON_SPACESIM_SLOWER_LINK
DIBUTTON_SPACESIM_TURN_LEFT_LINK
DIBUTTON_SPACESIM_TURN_RIGHT_LINK
DIBUTTON_SPACESIM_VIEW
DIHATSWITCH_SPACESIM_GLANCE

Priority 1 controls are those controls that, at the minimum, should be mapped to a device object if possible. Priority 2 controls are less critical to basic game operation and may be mapped as needed.

Game actions are assigned to action-mapping constant controls in an array of DIACTION types. Each **DIACTION** type contains at least one game action, its associated action-mapping constant, any necessary flags, and a friendly name for the action. The remaining **DIACTION** members are not used at this point. The following code sample shows the array of **DIACTION** types defined in the ActionMapper sample. Note that *CInputMapper* is a class used and defined by the ActionMapper sample to contain many of the standard action-mapping procedures. In this case it uses its own **AddAction** function to add each new **DIACTION** type to an array of them contained in a DIACTIONFORMAT type.

```
Dim m_mapper As New CInputMapper

With m_mapper
.AddAction INPUT_LEFTRIGHT_AXIS, DIAXIS_SPACESIM_LATERAL, 0, "Turn"
.AddAction INPUT_UPDOWN_AXIS, DIAXIS_SPACESIM_MOVE, 0, "Move"
.AddAction INPUT_FIREWEAPONS, DIBUTTON_SPACESIM_FIRE, 0, "Shoot"
.AddAction INPUT_ENABLESHIELD, DIBUTTON_SPACESIM_GEAR, 0, "Shield"
.AddAction INPUT_DISPLAYGAMEMENU,DIBUTTON_SPACESIM_DISPLAY,0,"Display"
.AddAction INPUT_QUITGAME, DIBUTTON_SPACESIM_MENU, 0, "Quit Game"
```

```
' Keyboard input mappings
.AddAction INPUT_FORWARDTHRUST, DIKEYBOARD_UP, 0, "Forward thrust"
.AddAction INPUT_REVERSETHRUST, DIKEYBOARD_DOWN, 0, "Reverse thrust"
.AddAction INPUT_FIREWEAPONS, DIKEYBOARD_F, 0, "Fire weapons"
.AddAction INPUT_ENABLESHIELD, DIKEYBOARD_S, 0, "Enable shields"
.AddAction INPUT_DISPLAYGAMEMENU, DIKEYBOARD_D, 0, "Display game menu"
.AddAction INPUT_QUITGAME, DIKEYBOARD_ESCAPE, 0, "Quit game"
.AddAction INPUT_TURNRIGHT, DIKEYBOARD_RIGHT, 0, "Right Turn"
.AddAction INPUT_TURNLEFT, DIKEYBOARD_LEFT, 0, "Left Turn"

' Mouse input mappings
.AddAction INPUT_LEFTRIGHT_AXIS, DIMOUSE_XAXIS, 0, "Turn"
.AddAction INPUT_UPDOWN_AXIS, DIMOUSE_YAXIS, 0, "Move"
.AddAction INPUT_FIREWEAPONS, DIMOUSE_BUTTON0, 0, "Fire weapons"
.AddAction INPUT_ENABLESHIELD, DIMOUSE_BUTTON1, 0, "Enable shields"
End With
```

The **DIACTIONFORMAT** type containing the array of **DIACTION** types will be used to find a device that can best be used with it. This is shown in Step 3: Mapping Actions to Devices.

Step 3: Mapping Actions to Devices

The ActionMapper sample uses its class function *CreateDevicesFromMap* to, among other things, measure the action map defined above against available devices. To do so, it first assigns values to the other members of the **DIACTIONFORMAT** type containing the array of **DIACTION** types. The values used in the sample have been passed as parameters to the **CreateDevicesFromMap** method. For clarity, the following code sample uses the actual values rather than the parameter names.

```
m_diaf.guidActionMap = "{20CAA014-60BC-4399-BDD3-84AD65A38A1C}"
m_diaf.lGenre = DIVIRTUAL_SPACESIM
m_diaf.lBufferSize = 16
m_diaf.lAxisMax = 100
m_diaf.lAxisMin = -100
m_diaf.ActionMapName = "Semantic Mapper VB Sample"
m_diaf.lActionCount = 18
```

The **guidActionMap** value is defined by the program. For more information on creating GUIDs, see Using GUIDS.

Once the desired mapping has been defined, it must be applied to physical devices. **DirectInput8.GetDevicesBySemantics** begins this process in the following code fragment, where *m_DI* is a previously defined **DirectInput8** object, *m_DIEnum* is a previously defined **DirectInputEnumDevices8** object, and *m_strUserName* is a previously defined string containing a friendly identifier for the user.

```
Set m_DIEnum = m_DI.GetDevicesBySemantics(m_strUserName, m_diaf, 0)
```

Flags could be used to limit the enumeration to a more specific search, but in this case no flags are set.

The results of the enumeration are examined individually and devices are created for each keyboard, mouse, and joystick found. In the following code sample, *devinst* is a previously defined **DirectInputDeviceInstance8** object, *m_Devices* is a previously defined array of **DirectInputDevice8** objects, and *m_DeviceTypes* is a previously defined array of **Long** values.

```
For i = 1 To m_DIEnum.GetCount

    Set devinst = m_DIEnum.GetItem(i)
    Set m_Devices(i) = m_DI.CreateDevice(devinst.GetGuidInstance)
    m_DeviceTypes(i) = devinst.GetDevType
    Set devinst = Nothing
```

The loop continues by checking whether the enumerated device currently under examination is a mouse. If so, the AXISMODE property is set to relative. Other properties could be set within this loop as well, but in the case of the ActionMapper sample, this is all that will be done here.

```
If m_DeviceTypes(i) = DI8DEVTYPE_MOUSE Then
   Dim dipl As DIPROPLONG
   dipl.lHow = DIPH_DEVICE
   dipl.lData = DIPROPAXISMODE_REL
   m_Devices(i).SetProperty "DIPROP_AXISMODE", dipl
End If
```

DirectInputDevice8.BuildActionMap takes the list of control assignments specified in the **DIACTIONFORMAT** type and attempts to map them to specific device objects on the device being enumerated for the user specified in the *m_strUserName* parameter. The results of the mapping are returned in the same structure. Each pass maps different controls depending on the nature of the enumerated device currently being examined. A keyboard, for instance, maps controls that specify a mapping to a particular key, but it leaves controls that specify buttons, sliders, or axes unmapped. If a joystick is enumerated next, **BuildActionMap** ignores those controls mapped to keyboard keys, but it attempts to map the buttons, sliders, and axes to the joystick's device objects.

```
m_Devices(i).BuildActionMap m_diaf, m_strUserName, 0
```

Once again, no flags are used in this instance.

The results of this default mapping can be examined and manipulated. This is done in Step 4: Configuring and Applying the Action Map.

Step 4: Configuring and Applying the Action Map

Once the mapping has been returned in the **DIACTIONFORMAT** type, it can be manipulated before being applied by **DirectInputDevice8.SetActionMap**.

The **lHow** member of each **DIACTION** type contained within the **DIACTIONFORMAT** type might be examined at this point to determine both whether the mapping was successful and what the criteria were in selecting the device object to which that action was mapped. For instance, you can determine if the mapping was specified by the hardware manufacturer or requested by the user.

You can change the actual mapping itself by editing the **lSemantic** member of the appropriate **DIACTION** structure returned in the default mapping. However, making these changes is not recommended.

Once the action-map configuration is finalized, it must be applied to the device to bind the physical controls to the game actions. This is done by calling the **DirectInputDevice8.SetActionMap** method as follows.

```
m_Devices(i).SetActionMap m_diaf, m_strUserName, 0
```

The parameters once again consist of the **DIACTIONFORMAT** type, the user name, and no flags.

Action maps, once created, can be displayed for and easily altered by the user. This is explained in Step 5: Displaying the Action Map.

Step 5: Displaying the Action Map

In addition to the advantage of action mapping in enabling the developer to define game actions independently of specific controller hardware, action mapping also benefits the user by providing a user interface (UI) for easy reconfiguration.

The UI can be displayed in two forms, text-only or graphics. In addition, each of those two options can be viewed in two modes: view-only or edit.

To display the UI, some preliminary setup is involved. A **DICONFIGUREDEVICESPARAMS** type is used to hold the property sheet information. **DICONFIGUREDEVICESPARAMS** contains an array of **DIACTIONFORMAT** types used for any mappings that have been defined, an array of user names to match to those action maps, and counts for both of the arrays. For this example, only one mapping and one user have been defined above. The initialization of the **DICONFIGUREDEVICESPARAMS** type is shown in the following sample code.

```
Dim m_cdParams As DICONFIGUREDEVICESPARAMS

    m_cdParams.ActionFormats(0) = m_diaf
```

```
m_cdParams.FormatCount = 1
m_cdParams.UserNames(0) = m_strUserName
m_cdParams.UserCount = 1
```

Once that structure is initialized, call **DirectInput8.ConfigureDevices** to display the UI as shown in the following sample code.

```
m_DI.ConfigureDevices 0, m_cdParams, DICD_EDIT
```

All that the sample is passing to **ConfigureDevices** is the **DICONFIGUREDEVICESPARAMS** type and a flag indicating that the UI allows user configuration. This is discussed in Step 6: User Configuration of the Action Map.

Step 6: User Configuration of the Action Map

At this point, the user configuration interface is displayed. If the device manufacturer has provided a graphic, users see an image of the device with labels indicating the action assigned to each device object. If no graphic is available, a simple text list of available device objects with their associated actions is displayed. An example of the former is shown here.

Users can now reassign controllers to actions as they see fit. Any changes made will be reflected in the **DIACTIONFORMAT** type. Once the user interface is closed, the new action map must be sent to **DirectInputDevice8.SetActionMap** before it can be implemented.

Using the data associated with the controls in an action map is described in Step 7: Retrieval of Action Mapped Data.

Step 7: Retrieval of Action Mapped Data

Action mapping was designed with buffered data in mind and will most often be used in that manner. The only difference between retrieving data from an action-mapped device and a device with no mapping is where you get it. Without action mapping, you would look to the **lOfs** member of the **DIDEVICEOBJECTDATA** type to determine which device object reported a change. However, with action mapping, you look to the **lUserData** member. Data is found as usual in the **lData** member.

DirectInput C/C++ Samples

The following sample programs demonstrate the use and capabilities of Microsoft® DirectInput®:

- DIConfig Sample
- FFConst Sample
- JoyStick Sample
- Keyboard Sample
- Mouse Sample
- MultiMapper Sample
- ReadFFE Sample
- Scrawl Sample

Although Microsoft DirectX® samples include Microsoft Visual C++® project workspace files, you might need to verify other settings in your development environment to ensure that the samples compile properly. For more information, see Compiling DirectX Samples and Other DirectX Applications.

DIConfig Sample

Description

The DIConfig code demonstrates the implementation of a configuration user interface based upon the Microsoft® DirectInput® Mapper technology. This sample code is very complex and is intended to be used more as a reference implementation than as a learning tool.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\DIConfig

Executable: None (see Programming Notes).

User's Guide

This sample does not create an executable file, so there are no end-user instructions.

Programming Notes

This code generates Diconfig.dll, a binary file that contains all of the functionality used in the default Mapper UI. This code is very complex and is intended as a reference implementation of a DirectInput-based configuration user interface.

For more information about this sample, see the comments within the sample code itself. The following major features are supported.

- Display of device images.
- Reconfiguration of devices.
- Support for multiple device views to illustrate alternate viewing angles.

- Support for control activation overlays.
- Use of the **GetImageInfo** method.
- Device ownership for multiuser applications on the same computer.
- Persistence of user settings through the **SetActionMap** method.

FFConst Sample

Description

The FFConst sample program applies raw forces to a force-feedback input device, illustrating how a simulator-type application can use force feedback to generate forces computed by a physics engine.

You must have a force-feedback device connected to your system in order to run the application.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\FFConst

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\Bin\FFConst.exe

User's Guide

When you run the application, it displays a window with a crosshair and a black spot in it. Click anywhere within the window's client area to move the black spot. (Note that moving the device itself does not do anything.) FFConst exerts a constant force on the device from the direction of the spot, in proportion to the distance from the crosshair. You can also hold down the mouse button and move the spot continuously.

Programming Notes

This sample program enumerates the input devices and acquires the first force-feedback device that it finds. If none are detected, it displays a message and terminates.

When the user moves the black spot, the **joySetForcesXY** function converts the cursor coordinates to a force direction and magnitude. This data is used to modify the parameters of the constant force effect.

Joystick Sample

Description

The Joystick sample program obtains and displays joystick data.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\Joystick

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\Bin\Joystick.exe

User's Guide

Observe how the displayed data changes when you move and twist the stick, rotate the throttle wheel, and press buttons in various combinations.

Programming Notes

The application polls the joystick for immediate data in response to a timer set inside the dialog procedure.

Keyboard Sample

Description

The Keyboard sample program enables the exploration of Foreground versus Background and Exclusive versus Non-Exclusive cooperative levels, and Buffered versus Immediate data retrieval from the keyboard.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\Keyboard

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\Bin\Keyboard.exe

User's Guide

Through the user interface, any combination of cooperative level and data styles can be created and the resulting keyboard behavior observed. Each setting is explained as it is selected, and the expected behavior is predicted when the device is created.

Programming Notes

Note that holding down a key in immediate mode holds that key indefinitely as subsequent snapshots of the keyboard are taken by Microsoft® DirectInput®. This is an optical illusion because DirectInput is taking individual snapshots at the rate of twelve times a second, but it demonstrates that, in immediate mode, only the state of the keyboard at that time is considered, while anything that has happened between snapshots is ignored.

Holding down a key in buffered mode briefly displays the data indicating that the key has been pressed, then quickly clears that data as it is flushed from the buffer.

Releasing the key generates another briefly displayed key-up event. This is a significant difference from immediate mode.

Mouse Sample

Description

The Mouse sample program enables the exploration of Foreground versus Background and Exclusive versus Non-Exclusive cooperative levels, and Buffered versus Immediate data retrieval from the mouse.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\Mouse

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\Bin\Mouse.exe

User's Guide

Through the user interface, any combination of cooperative level and data styles can be created and the resulting keyboard behavior observed. Each setting is explained as it is selected, and the expected behavior is predicted when the device is created.

Programming Notes

To release the mouse, press ENTER.

Note that in immediate mode, all device objects are visible and you can see the data change as the mouse is moved. This is an optical illusion because Microsoft® DirectInput® is taking individual snapshots at the rate of twelve times a second, but it demonstrates that, in immediate mode, only the state of the device at that time is considered, while anything that has happened between device snapshots is ignored.

Holding down a button in buffered mode briefly displays the data indicating that the button has been pressed, then quickly clears that data as it is flushed from the buffer. Releasing the button generates another briefly displayed button up event. This is a significant difference from immediate mode.

MultiMapper Sample

Description

The MultiMapper sample program shows how to set up, apply, and retrieve data using the Action Mapping functionality. It also demonstrates remapping through the user interface.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\MultiMapper

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\MultiMapper.exe

User's Guide

First, assign a number of players. The MultiMapper sample program accepts up to four. There must be one device, including the keyboard and the mouse, for each player. If more players are chosen than there are devices available, the program automatically scales back to the maximum number of players based on the devices attached.

The MultiMapper sample program is designed as though it were a space combat application. Each player is assigned a basic set of actions such as turning, firing, and activating thrusters. These actions are then mapped to device controls. Activation of those controls is then reflected by data in the display window.

To activate the action mapping user interface, press D. Each device has a separate tab for mapping that particular device. Double-clicking a device object label activates the list of possible actions to be assigned to that device object. Make a selection from the action list to map that action to the selected device object. You can also reassign devices among players.

Programming Notes

If a controller such as a joystick or a wheel is attached to the system, it displays a graphic of the device with actions as callouts to each device object. The display of this graphic depends on the presence of a provided bitmap for that device.

ReadFFE Sample

Description

The ReadFFE sample program enumerates and plays all of the Microsoft® DirectInput® Force Feedback effects stored in a DirectInput effects file created by the Force Editor.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\ReadFFE

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\ReadFFE.exe

Media: *(SDK root)*\Samples\Multimedia\Media*.ffe

User's Guide

Click **Read File** and open an existing Force Feedback Effects (FFE) file. If it was successfully read, click **Play Effects** to play the effects on the force-feedback device. FFE files can be authored using the Force Feedback Editor utility packaged with the Microsoft DirectX® SDK.

Programming Notes

For each file effect enumerated, the **EnumAndCreateEffectsCallback** function initializes an **IDirectInputEffect** pointer and adds it to a linked list. The **OnPlayEffects** function traverses this list and plays all effects.

Scrawl Sample

Description

The Scrawl application demonstrates using the mouse in exclusive mode in a windowed application. It functions in the same way as a pencil tool in a paint program.

Path

Source: *(SDK root)*\Samples\Multimedia\DirectInput\Scrawl

Executable: *(SDK root)*\Samples\Multimedia\DirectInput\Bin\Scrawl.exe

User's Guide

To draw, hold down the left mouse button and move the mouse. Click the right mouse button to invoke a shortcut menu. From the shortcut menu you can clear the client window, set the mouse sensitivity, or close the application. If the mouse buttons have been switched through Control Panel settings, Scrawl will detect and mimic this behavior. Note that Microsoft® DirectInput® ignores Control Panel settings and the application itself must make the switch.

Programming Notes

The Scrawl sample program demonstrates many aspects of DirectInput programming, including the following:

- Using the mouse in non-exclusive mode in a windowed application.
- Releasing the mouse when Microsoft Windows® needs to use it for menu access.
- Reacquiring the mouse when Windows no longer needs it.
- Reading buffered device data.
- Deferring screen updates until movement on both axes has been fully processed.

- Displaying event notifications of device activity.
- Restricting the cursor to an arbitrary region.
- Scaling raw mouse coordinates before using them.
- Using relative axis mode.

DirectInput Visual Basic Samples

The following sample programs demonstrate the use and capabilities of Microsoft® DirectInput®:

- ActionMapper Sample
- ForceFeedback Sample
- Joystick Sample
- Keyboard Sample
- ScrawlB Sample

ActionMapper Sample

Description

The ActionMapper sample program demonstrates the retrieval of action-mapped data. It also enables reconfiguration of the keyboard, mouse, and other controller action mappings through the user interface.

Path

Source: *(SDK root)*\Samples\Multimedia\VBSamples\DirectInput\ActionMapper

Executable: Source: *(SDK root)* \Samples\Multimedia\VBSamples\DirectInput\Bin\VB_ActionMapper.exe

User's Guide

The initial display gives a text feedback of axis movements and button or key presses from the mouse, keyboard, or other controller attached to the system. An EXCLUSIVE | FOREGROUND cooperative level is used so that the mouse cursor is hidden.

Pressing the D key displays the configuration user interface with tabs for the mouse, keyboard, and any other present controller device. If a graphic is available for the controller, it is displayed with callouts indicating which action is associated with

which control. Double-clicking a callout label activates the menu of possible actions to apply to that control. To return to the initial screen with the new mappings, click OK.

To terminate the program, press the ESC key from the initial screen.

Programming Notes

You will need to modify ActionMap.cls. For the purpose of this sample, it queries for any and all input devices, and it does not distinguish the source of the input. However, you will usually modify the class to respond to only one given input, or to differentiate the input devices into different players or purposes.

This sample makes use of common DirectX code that is shared with other samples on the DirectX SDK. All common classes and modules can be found in the following directory:

(SDK root)\Samples\Multimedia\VBSamples\Common

ForceFeedback Sample

Description

The ForceFeedback sample program is a simple force-feedback editor that enables you to create an effect, set parameters, and play the effect.

Path

Source: *(SDK root)*\Samples\Multimedia\VBSamples\DirectInput\Feedback

Executable: *(SDK root)*
\Samples\Multimedia\VBSamples\DirectInput\Bin\VB_ForceFeedback.exe

User's Guide

Select an effect type from the list, and modify the parameters. Play the effect by pressing the primary button on the joystick.

Programming Notes

The ForceFeedback sample program shows how to turn off the autocenter property and how to create and modify effects. Changes in effect parameters are applied immediately.

Joystick Sample

Description

The Joystick sample program shows how to enumerate joysticks, set joystick properties, and get immediate data.

Path

Source: *(SDK root)*\Samples\Multimedia\VBSamples\DirectInput\Joystick

Executable: *(SDK root)*
\Samples\Multimedia\VBSamples\DirectInput\Bin\VB_Joystick.exe

User's Guide

Select a joystick from the list. The data from the axes and buttons is displayed. Observe how the displayed data changes when you move and twist the stick, rotate the throttle wheel, and press buttons in various combinations.

Programming Notes

The list box is unsorted so that the selection can easily be retrieved from the **DirectInputEnumDevices8** collection.

Keyboard Sample

Description

The Keyboard sample program shows how to create a keyboard device and retrieve immediate data from it.

Path

Source: *(SDK root)*\Samples\Multimedia\VBSamples\Dinput\Src\Keyboard

Executable: *(SDK root)*\Samples\Multimedia\VBSamples\Dinput\Bin\Keyboard.exe

User's Guide

As you press one or more keys while the application has focus, the names of the keys (from **CONST_DIKEYFLAGS**) are displayed in the list box.

Programming Notes

The background cooperative level is used, so keys are read even when the application does not have the input focus.

ScrawlB Sample

Description

The ScrawlB sample program illustrates using Microsoft® DirectInput® to create a simple drawing program. It shows how to use the mouse in exclusive mode, how to use event notification for mouse events, how to retrieve and interpret buffered data, and how to use callback functions to receive mouse movement events.

Path

Source: *(SDK root)*\Samples\Multimedia\VBSamples\DirectInput\Scrawl

Executable: *(SDK root)*
\Samples\Multimedia\VBSamples\DirectInput\Bin\VB_Scrawlb.exe

User's Guide

Hold down the left button while dragging the mouse to draw. Click the right button to see a shortcut menu. Choosing **Release Mouse** (or pressing and releasing ALT) releases the system cursor so that you can move or resize the window or click on another application. Click on the client window again to resume drawing.

Programming Notes

The application subclasses the Display window to capture WM_ENTERMENULOOP messages. This is the best way to find out if the user has opened the system menu by pressing ALT+SPACEBAR, so that ScrawlB can release the mouse through a call to SetWindowLong. However, subclassing can lead to difficulties when debugging. Note that failure to comment out these lines while running the sample from within the Visual Basic environment will result in undefined behavior.

The *frmCanvas* module implements **DirectXEvent8** in order to be able to process input events. The Microsoft® Visual Basic® **MouseMove** event is also used by this form for getting Microsoft Windows® mouse events when Microsoft DirectInput® does not have access to the mouse. When this event is triggered, the application knows that the cursor is back in the client window, so the mouse can be reacquired.

DirectInput C/C++ Reference

Reference material for the Microsoft® DirectInput® C/C++ application programming interface (API) is divided into the following categories.

- Interfaces

- Callback Functions
- Functions
- Macros
- Structures
- Device Constants
- Action Mapping Constants
- Return Values

Interfaces

This section contains references for methods of the following Microsoft® DirectInput® interfaces.

- **IDirectInput8**
- **IDirectInputDevice8**
- **IDirectInputEffect**

Note

All DirectInput methods have corresponding macros that expand to C or C++ syntax depending on which language is defined. These macros are found in the Dinput.h header file and are not documented separately.

IDirectInput8

Applications use the methods of the **IDirectInput8** interface to enumerate, create, and retrieve the status of Microsoft® DirectInput® devices, initialize the DirectInput object, and invoke an instance of the Microsoft Windows® Control Panel.

IDirectInput8 supersedes the **IDirectInput**, **IDirectInput2**, and **IDirectInput7** interfaces used in earlier versions of Microsoft DirectX®.

IDirectInput8 is an interface to a new class of object, represented by the class identifier CLSID_DirectInput8, and cannot be obtained by calling QueryInterface on an interface to objects of class CLSID_DirectInput. Instead, obtain the **IDirectInput8** interface by using the **DirectInput8Create** function.

The methods of the **IDirectInput8** interface can be organized into the following groups.

Device Management	**ConfigureDevices**
	CreateDevice
	EnumDevices
	EnumDevicesBySemantics
	FindDevice

	GetDeviceStatus
Miscellaneous	**Initialize**
	RunControlPanel

The **IDirectInput** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

IUnknown	**AddRef**
	QueryInterface
	Release

The **LPDIRECTINPUT8** type is defined as a pointer to the **IDirectInput8** interface:

```
typedef struct IDirectInput8    *LPDIRECTINPUT8;
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInput8::ConfigureDevices

Displays property pages for connected input devices and enables the user to map actions to device controls.

```
HRESULT IDirectInput8::ConfigureDevices(
  LPDICONFIGUREDEVICESCALLBACK lpdiCallback,
  LPDICONFIGUREDEVICESPARAMS   lpdiCDParams,
  DWORD  dwFlags,
  LPVOID pvRefData
);
```

Parameters

lpdiCallback

Address of a callback function to be called each time the contents of the surface change. See **DIConfigureDevicesCallback**. Pass NULL if the application does not handle the display of the property sheet. In this case, Microsoft® DirectInput® displays the property sheet and returns control to the application when the user closes the property sheet. If you supply a callback pointer, you must also supply a valid surface pointer in the **lpUnkDDSTarget** member of the **DICONFIGUREDEVICESPARAMS** structure.

lpdiCDParams

Address of a **DICONFIGUREDEVICESPARAMS** structure that contains information about users and genres for the game, as well as information about how the user interface is displayed.

dwFlags

DWORD value that specifies the mode in which the control panel should be invoked. *DwFlags* must be one of the following values.

DICD_DEFAULT

Open the property sheet in view-only mode.

DICD_EDIT

Open the property sheet in edit mode. This mode enables the user to change action-to-control mappings. After the call returns, the application should assume current devices are no longer valid, release all device interfaces, and reinitialize them by calling **IDirectInput8::EnumDevicesBySemantics**.

pvRefData

Application-defined value to pass to the callback function.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following.

DIERR_INVALIDPARAM

DIERR_OUTOFMEMORY

Remarks

Hardware vendors provide bitmaps and other display information for their device.

Before calling the method, an application can modify the text labels associated with each action by changing the value in the **lptszActionName** member of the **DIACTION** structure.

Configuration is stored for each user of each device for each game. The information can be retrieved by the **IDirectInputDevice8::BuildActionMap** method.

By default, acceleration is supported for these pixel formats:

A1R5G5B5

16-bit pixel format with 5 bits reserved for each color and 1 bit reserved for alpha (transparent texel).

A8R8G8B8

32-bit ARGB pixel format with alpha.

R9G8B8

24-bit RGB pixel format.

X1R5G5B5

16-bit pixel format with 5 bits reserved for each color.

X8R8G8B8
32-bit RGB pixel format with 8 bits reserved for each color.

Other formats will result in color conversion and dramatically slow the frame rate.

Note

Even if the cooperative level for the application is disabling the Microsoft Windows® logo key passively through an exclusive cooperative level or actively through use of the DISCL_NOWINKEY flag, that key will be active while the default action mapping UI is displayed.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInput8::CreateDevice

Creates and initializes an instance of a device based on a given GUID, and obtains an **IDirectInputDevice8** interface.

```
HRESULT CreateDevice(
  REFGUID rguid,
  LPDIRECTINPUTDEVICE *lplpDirectInputDevice,
  LPUNKNOWN pUnkOuter
);
```

Parameters

rguid
Reference to (C++) or address of (C) the instance GUID for the desired input device (see Remarks). The GUID is retrieved through the **IDirectInput8::EnumDevices** method, or it can be one of the following predefined GUIDs:

GUID_SysKeyboard
The default system keyboard.

GUID_SysMouse
The default system mouse.

For the preceding GUID values to be valid, your application must define INITGUID before all other preprocessor directives at the beginning of the source file, or link to Dxguid.lib.

lplpDirectInputDevice
Address of a variable to receive the **IDirectInputDevice8** interface pointer if successful.

pUnkOuter

Address of the controlling object's **IUnknown** interface for COM aggregation, or NULL if the interface is not aggregated. Most callers pass NULL.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following:

DIERR_DEVICENOTREG
DIERR_INVALIDPARAM
DIERR_NOINTERFACE
DIERR_NOTINITIALIZED
DIERR_OUTOFMEMORY

Remarks

Calling this method with *pUnkOuter* = NULL is equivalent to creating the object by **CoCreateInstance**(*&CLSID_DirectInputDevice, NULL, CLSCTX_INPROC_SERVER, riid, lplpDirectInputDevice*) and then initializing it with **Initialize**.

Calling this method with *pUnkOuter* != NULL is equivalent to creating the object by **CoCreateInstance**(&CLSID_DirectInputDevice, punkOuter, CLSCTX_INPROC_SERVER, &IID_IUnknown, lplpDirectInputDevice). The aggregated object must be initialized manually.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInput8::EnumDevices

Enumerates available devices.

```
HRESULT EnumDevices(
  DWORD dwDevType,
  LPDIENUMCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

dwDevType

Device type filter.

To restrict the enumeration to a particular type of device, set this parameter to a DI8DEVTYPE_* value. See **DIDEVICEINSTANCE**.

To enumerate a class of devices, use one of the following values.

DI8DEVCLASS_ALL
All devices.

DI8DEVCLASS_DEVICE
All devices that do not fall into another class.

DI8DEVCLASS_GAMECTRL
All game controllers.

DI8DEVCLASS_KEYBOARD
All keyboards. Equivalent to DI8DEVTYPE_KEYBOARD.

DI8DEVCLASS_POINTER
All devices of type DI8DEVTYPE_MOUSE and DI8DEVTYPE_SCREENPOINTER.

lpCallback

Address of a callback function to be called once for each device enumerated. See **DIEnumDevicesCallback**.

pvRef

Application-defined 32-bit value to be passed to the enumeration callback each time it is called.

dwFlags

Flag value that specifies the scope of the enumeration. This parameter can be one or more of the following values:

DIEDFL_ALLDEVICES
All installed devices are enumerated. This is the default behavior.

DIEDFL_ATTACHEDONLY
Only attached and installed devices.

DIEDFL_FORCEFEEDBACK
Only devices that support force feedback.

DIEDFL_INCLUDEALIASES
Include devices that are aliases for other devices.

DIEDFL_INCLUDEHIDDEN
Include hidden devices. For more information about hidden devices, see **DIDEVCAPS**.

DIEDFL_INCLUDEPHANTOMS
Include phantom (placeholder) devices.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

All installed devices can be enumerated, even if they are not present. For example, a flight stick might be installed on the system but not currently plugged into the computer. Set the *dwFlags* parameter to indicate whether only attached or all installed devices should be enumerated. If the DIEDFL_ATTACHEDONLY flag is not present, all installed devices are enumerated.

A preferred device type can be passed as a *dwDevType* filter so that only the devices of that type are enumerated.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInput8::EnumDevicesBySemantics

IDirectInput8::EnumDevicesBySemantics

Enumerates devices that most closely match the application-specified action map.

```
HRESULT EnumDevicesBySemantics(
  LPCTSTR ptszUserName,
  LPDIACTIONFORMAT lpdiActionFormat,
  LPDIENUMDEVICESBYSEMANTICSCB lpCallback,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

ptszUserName
String identifying the current user, or NULL to specify the user logged onto the system. The user name is taken into account when enumerating devices. A device

with user mappings is preferred to a device without any user mappings. By default, devices in use by other users are not enumerated for this user.

lpdiActionFormat
Address of a **DIACTIONFORMAT** structure that specifies the action map for which suitable devices are enumerated.

lpCallback
Address of a callback function to be called once for each device enumerated. See **DIEnumDevicesBySemanticsCallback**.

pvRef
Application-defined 32-bit value to pass to the enumeration callback each time it is called.

dwFlags
Flag value that specifies the scope of the enumeration. This parameter can be one or more of the following values.

DIEDBSFL_ATTACHEDONLY
Only attached and installed devices are enumerated.

DIEDBSFL_AVAILABLEDEVICES
Only unowned, installed devices are enumerated.

DIEDBSFL_FORCEFEEDBACK
Only devices that support force feedback are enumerated.

DIEDBSFL_MULTIMICEKEYBOARDS
Only secondary (non-system) keyboard and mouse devices.

DIEDBSFL_NONGAMINGDEVICES
Only HID-compliant devices whose primary purpose is not as a gaming device. Devices such as USB speakers and multimedia buttons on some keyboards would fall within this value.

DIEDBSFL_THISUSER
All installed devices for the user identified by *ptszUserName*, and all unowned devices, are enumerated.

DIEDBSFL_VALID is also defined in Dinput.h, but is not used by applications.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values.

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

The keyboard and mouse are enumerated last.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInput8::EnumDevices, Action Mapping.

IDirectInput8::FindDevice

Retrieves the instance GUID of a device that has been newly attached to the system. It is called in response to a Microsoft® Win32® device management notification.

```
HRESULT FindDevice(
  REFGUID rguidClass,
  LPCTSTR ptszName,
  LPGUID pguidInstance
);
```

Parameters

rguidClass
Unique identifier of the device class for the device that the application is to locate. The application obtains the class GUID from the device arrival notification. For more information, see the documentation on the DBT_DEVICEARRIVAL event in the Microsoft Platform Software Development Kit (SDK).

ptszName
Name of the device. The application obtains the name from the device arrival notification.

pguidInstance
Address of a variable to receive the instance GUID for the device, if the device is found. This value can be passed to IDirectInput8::CreateDevice.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be DIERR_DEVICENOTREG. Failure results if the GUID and name do not correspond to a device class that is registered with Microsoft® DirectInput®. For example, they might refer to a storage device, rather than an input device.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInput8::GetDeviceStatus

Retrieves the status of a specified device.

```
HRESULT GetDeviceStatus(
  REFGUID rguidInstance
);
```

Parameters

rguidInstance
Reference to (C++) or address of (C) the GUID identifying the instance of the device whose status is being checked.

Return Values

If the method succeeds, the return value is DI_OK if the device is attached to the system, or DI_NOTATTACHED otherwise.

If the method fails, the return value can be one of the following error values:

DIERR_GENERIC
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInput8::Initialize

Initializes a Microsoft® DirectInput® object. Applications normally do not need to call this method. The **DirectInput8Create** function automatically initializes the DirectInput object after creating it.

```
HRESULT Initialize(
  HINSTANCE hinst,
```

DWORD *dwVersion*
);

Parameters

hinst

Instance handle to the application or dynamic-link library (DLL) that is creating the DirectInput object. DirectInput uses this value to determine whether the application or DLL has been certified and to establish any special behaviors that might be necessary for backwards compatibility.

It is an error for a DLL to pass the handle of the parent application. For example, a Microsoft ActiveX® control embedded in a Web page that uses DirectInput must pass its own instance handle, and not the handle of the Web browser. This ensures that DirectInput recognizes the control and can enable any special behaviors that might be necessary.

dwVersion

Version number of DirectInput for which the application is designed. This value is normally DIRECTINPUT_VERSION. Passing the version number of a previous version causes DirectInput to emulate that version.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_BETADIRECTINPUTVERSION
DIERR_OLDDIRECTINPUTVERSION

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInput8::RunControlPanel

Runs Control Panel to enable the user to install a new input device or modify configurations.

HRESULT RunControlPanel(
HWND *hwndOwner*,
DWORD *dwFlags*
);

Parameters

hwndOwner
Handle of the window to be used as the parent window for the subsequent user interface. If this parameter is NULL, no parent window is used.

dwFlags
Currently not used and must be set to 0.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::RunControlPanel

IDirectInputDevice8

Applications use the methods of the **IDirectInputDevice8** interface to gain and release access to Microsoft® DirectInput® devices, manage device properties and information, set behavior, perform initialization, create and play force-feedback effects, and invoke a device's control panel.

The **IDirectInputDevice8** interface is obtained by using the **IDirectInput8::CreateDevice** method. For an example, see Creating a DirectInput Device.

IDirectInputDevice8 supersedes the **IDirectInputDevice, IDirectInputDevice2,** and **IDirectInputDevice7** interfaces used in previous versions of Microsoft® DirectX®, but does not inherit from them. Methods that share names with those from older interfaces perform similar services, but may not have exactly the same functionality or behavior. You cannot obtain the earlier interfaces by using **QueryInterface**.

The methods of the **IDirectInputDevice8** interface can be organized into the following groups.

Accessing input devices	**Acquire**
	Unacquire
Action mapping	**BuildActionMap**
	GetImageInfo
	SetActionMap
Device information	**GetCapabilities**
	GetDeviceData
	GetDeviceInfo
	GetDeviceState
	Poll
	SetDataFormat
	SetEventNotification
Device objects	**EnumObjects**
	GetObjectInfo
Device properties	**GetProperty**
	SetCooperativeLevel
	SetProperty
Force feedback	**CreateEffect**
	EnumCreatedEffectObjects
	EnumEffects
	EnumEffectsInFile
	Escape
	GetEffectInfo
	GetForceFeedbackState
	SendForceFeedbackCommand
	WriteEffectToFile
Miscellaneous	**Initialize**
	RunControlPanel
	SendDeviceData

The **IDirectInputDevice8** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

IUnknown	**AddRef**
	QueryInterface
	Release

The **LPDIRECTINPUTDEVICE8** type is defined as a pointer to the **IDirectInputDevice8** interface:

```
typedef struct IDirectInputDevice8    *LPDIRECTINPUTDEVICE8;
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8

IDirectInputDevice8::Acquire

Obtains access to the input device.

```
HRESULT Acquire();
```

Parameters

None.

Return Values

If the method succeeds, the return value is DI_OK, or S_FALSE if the device was already acquired.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED
DIERR_OTHERAPPHASPRIO

Remarks

Before a device can be acquired, a data format must be set by using the **IDirectInputDevice8::SetDataFormat** method or **IDirectInputDevice8::SetActionMap** method. If the data format has not been set, **Acquire** returns DIERR_INVALIDPARAM.

Devices must be acquired before calling the **IDirectInputDevice8::GetDeviceState** or **IDirectInputDevice8::GetDeviceData** methods for that device.

Device acquisition does not use a reference count. Therefore, if an application calls the **IDirectInputDevice8::Acquire** method twice, then calls the **IDirectInputDevice8::Unacquire** method once, the device is unacquired.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::BuildActionMap

Builds an action map for the device and retrieves information about it.

```
HRESULT BuildActionMap(
  LPDIACTIONFORMAT lpdiaf,
  LPCWSTR lpszUserName
  DWORD  dwFlags
);
```

Parameters

lpdiaf
Address of a **DIACTIONFORMAT** structure that receives information about the action map.

lpszUserName
Pointer to a string that specifies the name of the user for whom mapping is requested. If NULL, the current user is assumed.

dwFlags
Flags to control the mapping. This can be one of the following values.

DIDBAM_DEFAULT
Overwrite all mappings except application-specified mappings; that is, mappings that have the DIA_APPMAPPED flag in the **DIACTION** structure.

DIDBAM_HWDEFAULTS
Overwrite all mappings, including application-specified mappings. This flag is similar to DIDBAM_INITIALIZE, but automatically overrides user-mapped actions with the defaults specified by the device driver or Microsoft® DirectInput®.

DIDBAM_INITIALIZE
Overwrite all mappings, including application-specified mappings.

DIDBAM_PRESERVE
Preserve current mappings assigned for this device or any other configured device.

Return Values

If the method succeeds, the return value is DI_OK, DI_NOEFFECT, or DI_WRITEPROTECT. See Remarks.

If the method fails, the return value can be one of the following error values.

DIERR_INVALIDPARAM
DIERR_MAPFILEFAIL

Remarks

The method returns DI_NOEFFECT if no mappings were created for the device. For example, a keyboard or mouse will not provide mappings for genre-specific actions.

If DIERR_INVALIDPARAM is returned, one or more of the mappings was not valid. The **dwHow** member of the **DIACTION** structure is set to DIAH_ERROR. The application can iterate through the action map to find and correct errors.

If DIEFF_MAPFILEFAIL is returned, an error has occurred either reading the vendor supplied file for the device or reading or writing the user configuration file for the device.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::SetActionMap, **IDirectInputDevice8::SetDataFormat**

IDirectInputDevice8::CreateEffect

Creates and initializes an instance of an effect identified by the effect GUID.

```
HRESULT CreateEffect(
  REFGUID rguid,
  LPCDIEFFECT lpeff,
  LPDIRECTINPUTEFFECT *ppdeff,
  LPUNKNOWN punkOuter
);
```

Parameters

rguid

Reference to (C++) or address of (C) the GUID identifying the effect to be created. This can be a predefined effect GUID, or it can be a GUID obtained from **IDirectInputDevice8::EnumEffects**.

The following standard effect GUIDs are defined:

- GUID_ConstantForce

- GUID_RampForce
- GUID_Square
- GUID_Sine
- GUID_Triangle
- GUID_SawtoothUp
- GUID_SawtoothDown
- GUID_Spring
- GUID_Damper
- GUID_Inertia
- GUID_Friction
- GUID_CustomForce

lpeff
DIEFFECT structure that provides parameters for the created effect. This parameter is optional. If it is NULL, the effect object is created without parameters. The application must then call the **IDirectInputEffect::SetParameters** method to set the parameters of the effect before it can download the effect.

ppdeff
Address of a variable to receive a pointer to the **IDirectInputEffect** interface if successful.

punkOuter
Controlling unknown for COM aggregation. The value is NULL if the interface is not aggregated. Most callers pass NULL.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_DEVICEFULL
DIERR_DEVICENOTREG
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

If the return value is S_OK, the effect was created, and the parameters of the effect were updated, but the effect was not necessarily downloaded. For it to be downloaded, the device must be acquired in exclusive mode.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for

Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::EnumCreatedEffectObjects

Enumerates all the currently created effects for this device. Effects created by **IDirectInputDevice8::CreateEffect** are enumerated.

```
HRESULT EnumCreatedEffectObjects(
  LPDIENUMCREATEDEFFECTOBJECTSCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD fl
);
```

Parameters

lpCallback
Address of an application-defined callback function. Microsoft® DirectInput® provides the prototype function **DIEnumCreatedEffectObjectsCallback**.

pvRef
Reference data (context) for callback.

fl
No flags are currently defined. This parameter must be 0.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

The results are unpredictable if you create or destroy an effect while an enumeration is in progress. However, the callback function can safely release the effect passed to it.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::EnumEffects

Enumerates all the effects supported by the force-feedback system on the device. The enumerated GUIDs can represent predefined effects, as well as effects peculiar to the device manufacturer.

```
HRESULT EnumEffects(
  LPDIENUMEFFECTSCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD dwEffType
);
```

Parameters

lpCallback
Address of an application-defined callback function. The declaration of this function must conform to that of the **DIEnumEffectsCallback** prototype.

pvRef
A 32-bit application-defined value to be passed to the callback function. This parameter can be any 32-bit type; it is declared as **LPVOID** for convenience.

dwEffType
Effect type filter. Use one of the DIEFT_* values to indicate the effect type to be enumerated, or DIEFT_ALL to enumerate all effect types. For a list of these values, see **DIEFFECTINFO**.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

If the callback stops the enumeration prematurely, the enumeration is considered to have succeeded.

Remarks

An application can use the **dwEffType** member of the **DIEFFECTINFO** structure to obtain general information about the effect, such as its type and which envelope and condition parameters are supported by the effect.

To exploit an effect to its fullest, contact the device manufacturer to obtain information on the semantics of the effect and its effect-specific parameters.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::EnumEffectsInFile

Enumerates all the effects in a file created by the Force Editor utility or another application using the same file format.

```
HRESULT EnumEffectsInFile(
  LPCSTR lpszFileName,
  LPENUMEFFECTSINFILECALLBACK pec,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

lpszFileName
Name of the RIFF file.

pec
Address of an application-defined callback function. The declaration of this function must conform to that of the **DIEnumEffectsInFileCallback** prototype.

pvRef
Application-defined value to be passed to the callback function. This parameter can be any 32-bit type.

dwFlags
Can be DIFEF_DEFAULT (= 0) or one or both of the following values:

DIFEF_INCLUDENONSTANDARD
Include effect types that are not defined by Microsoft® DirectInput®.

DIFEF_MODIFYIFNEEDED
Instruct DirectInput to modify the authored effect, if necessary, so that it plays on the current device. For example, by default, an effect authored for two axes does not play on a single-axis device. Setting this flag allows the effect to play on a single axis. The parameters are modified in the **DIEFFECT** structure pointed to by the **lpDiEffect** member of the **DIFILEEFFECT** structure passed to the callback.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

If the callback stops the enumeration prematurely, the enumeration is considered to have succeeded.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::WriteEffectToFile, Loading Effects from a File

IDirectInputDevice8::EnumObjects

Enumerates the input and output objects available on a device.

```
HRESULT EnumObjects(
  LPDIENUMDEVICEOBJECTSCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

lpCallback
Address of a callback function that receives DirectInputDevice objects. Microsoft® DirectInput® provides a prototype of this function as **DIEnumDeviceObjectsCallback**.

pvRef
Reference data (context) for callback.

dwFlags
Flags that specify the types of object to be enumerated. Each of the following values restricts the enumeration to objects of the described type:

DIDFT_ABSAXIS
An absolute axis.

DIDFT_ALIAS
Controls identified by a Human Interface Device usage alias. This flag applies only to HID-compliant USB devices.

DIDFT_ALL
All objects.

DIDFT_AXIS
An axis, either absolute or relative.

DIDFT_BUTTON
A push button or a toggle button.

DIDFT_COLLECTION
A Human Interface Device (HID) link collection. HID link collections do not generate data of their own.

DIDFT_ENUMCOLLECTION(*n*)
An object that belongs to HID link collection number *n*.

DIDFT_FFACTUATOR
An object that contains a force-feedback actuator. In other words, forces can be applied to this object.

DIDFT_FFEFFECTTRIGGER
An object that can be used to trigger force-feedback effects.

DIDFT_NOCOLLECTION
An object that does not belong to any HID link collection; in other words, an object for which the **wCollectionNumber** member of the **DIDEVICEOBJECTINSTANCE** structure is 0.

DIDFT_NODATA
An object that does not generate data.

DIDFT_OUTPUT
An object to which data can be sent by using the **IDirectInputDevice8::SendDeviceData** method.

DIDFT_POV
A point-of-view controller.

DIDFT_PSHBUTTON
A push button. A push button is reported as down when the user presses it and as up when the user releases it.

DIDFT_RELAXIS
A relative axis.

DIDFT_TGLBUTTON
A toggle button. A toggle button is reported as down when the user presses it and remains so until the user presses the button a second time.

DIDFT_VENDORDEFINED
An object of a type defined by the manufacturer.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

The DIDFT_FFACTUATOR and DIDFT_FFEFFECTTRIGGER flags in the *dwFlags* parameter restrict enumeration to objects that meet all the criteria defined by the included flags. For all the other flags, an object is enumerated if it meets the criterion defined by any included flag in this category. For example, (DIDFT_FFACTUATOR | DIDFT_FFEFFECTTRIGGER) restricts enumeration to force-feedback trigger objects, and (DIDFT_FFEFFECTTRIGGER | DIDFT_TGLBUTTON | DIDFT_PSHBUTTON) restricts enumeration to buttons of any kind that can be used as effect triggers.

Applications should not rely on this method to determine whether certain keyboard keys or indicator lights are present, as these objects might be enumerated even though they are not present. Although the basic set of available objects can be determined from the device subtype, there is no reliable way of determining whether extra objects such as the menu key are available.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::Escape

Sends a hardware-specific command to the force-feedback driver.

```
HRESULT Escape(
  LPDIEFFESCAPE pesc
);
```

Parameters

pesc

DIEFFESCAPE structure that describes the command to be sent. On success, the **cbOutBuffer** member contains the number of bytes of the output buffer actually used.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_DEVICEFULL
DIERR_NOTINITIALIZED

Other device-specific error codes are also possible. Ask the hardware manufacturer for details.

Remarks

Because each driver implements different escapes, it is the application's responsibility to ensure that it is sending the escape to the correct driver by comparing the value of the **guidFFDriver** member of the **DIDEVICEINSTANCE** structure against the value the application is expecting.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetCapabilities

Obtains the capabilities of the DirectInputDevice object.

```
HRESULT GetCapabilities(
  LPDIDEVCAPS lpDIDevCaps
);
```

Parameters

lpDIDevCaps
Address of a **DIDEVCAPS** structure to be filled with the device capabilities. The **dwSize** member of this structure must be initialized before calling this method.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED
E_POINTER

Remarks

For compatibility with Microsoft® DirectX® 3, it is also valid to pass a **DIDEVCAPS_DX3** structure with the **dwSize** member initialized to **sizeof(DIDEVCAPS_DX3)**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetDeviceData

Retrieves buffered data from the device.

```
HRESULT GetDeviceData(
  DWORD cbObjectData,
  LPDIDEVICEOBJECTDATA rgdod,
  LPDWORD pdwInOut,
  DWORD dwFlags
);
```

Parameters

cbObjectData
Size of the **DIDEVICEOBJECTDATA** structure, in bytes.

rgdod
Array of **DIDEVICEOBJECTDATA** structures to receive the buffered data. The number of elements in this array must be equal to the value of the *pdwInOut* parameter. If this parameter is NULL, the buffered data is not stored anywhere, but all other side-effects take place.

pdwInOut
On entry, the number of elements in the array pointed to by the *rgdod* parameter. On exit, the number of elements actually obtained.

dwFlags
Flags that control the manner in which data is obtained. This value can be 0 or the following flag:

DIGDD_PEEK
Do not remove the items from the buffer. A subsequent **IDirectInputDevice8::GetDeviceData** call will read the same data. Normally, data is removed from the buffer after it is read.

Return Values

If the method succeeds, the return value is DI_OK or DI_BUFFEROVERFLOW.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST
DIERR_INVALIDPARAM

DIERR_NOTACQUIRED
DIERR_NOTBUFFERED
DIERR_NOTINITIALIZED

Remarks

Before device data can be obtained, you must set the data format and the buffer size by using the **IDirectInputDevice8::SetDataFormat** and **IDirectInputDevice8::SetProperty** methods, or by using the **IDirectInputDevice8::SetActionMap** method. You must also acquire the device by using the **IDirectInputDevice8::Acquire** method.

The following code example reads up to ten buffered data elements, removing them from the device buffer as they are read.

```
DIDEVICEOBJECTDATA rgdod[10];
DWORD dwItems = 10;
hres = IDirectInputDevice8_GetDeviceData(
   sizeof(DIDEVICEOBJECTDATA),
   rgdod,
   &dwItems,
   0);
if (SUCCEEDED(hres)) {
   // dwItems = number of elements read (could be zero)
   if (hres == DI_BUFFEROVERFLOW) {
   // Buffer had overflowed.
   }
}
```

Your application can flush the buffer and retrieve the number of flushed items by specifying NULL for the *rgdod* parameter and a pointer to a variable containing INFINITE for the *pdwInOut* parameter. The following code example illustrates how this can be done:

```
dwItems = INFINITE;
hres = IDirectInputDevice8_GetDeviceData(
          pdid,
          sizeof(DIDEVICEOBJECTDATA),
          NULL,
          &dwItems,
          0);
if (SUCCEEDED(hres)) {
   // Buffer successfully flushed
   // dwItems = number of elements flushed
   if (hres == DI_BUFFEROVERFLOW) {
      // Buffer had overflowed.
   }
```

```
}
```

Your application can query for the number of elements in the device buffer by setting the *rgdod* parameter to NULL, setting *pdwInOut* to INFINITE and setting *dwFlags* to DIGDD_PEEK. The following code example illustrates how this can be done:

```
dwItems = INFINITE;
hres = IDirectInputDevice8_GetDeviceData(
        pdid,
        sizeof(DIDEVICEOBJECTDATA),
        NULL,
        &dwItems,
        DIGDD_PEEK);
if (SUCCEEDED(hres)) {
   // dwItems = number of elements in buffer
   if (hres == DI_BUFFEROVERFLOW) {
      // Buffer overflow occurred; not all data
      // was successfully captured.
   }
}
```

To query about whether a buffer overflow has occurred, set the *rgdod* parameter to NULL and the *pdwInOut* parameter to 0. The following code example illustrates how this can be done:

```
dwItems = 0;
hres = IDirectInputDevice8_GetDeviceData(
        pdid,
        sizeof(DIDEVICEOBJECTDATA),
        NULL,
        &dwItems,
        0);
if (hres == DI_BUFFEROVERFLOW) {
   // Buffer overflow occurred
}
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::Poll, Polling and Event Notification

IDirectInputDevice8::GetDeviceInfo

Obtains information about the device's identity.

```
HRESULT GetDeviceInfo(
  LPDIDEVICEINSTANCE pdidi
);
```

Parameters

pdidi

Address of a **DIDEVICEINSTANCE** structure to be filled with information about the device's identity. An application must initialize the structure's **dwSize** member before calling this method.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM

DIERR_NOTINITIALIZED

E_POINTER

Remarks

For compatibility with Microsoft® DirectX 3®, it is also valid to pass a **DIDEVICEINSTANCE_DX3** structure with the **dwSize** member initialized to **sizeof(DIDEVICEINSTANCE_DX3)**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetDeviceState

Retrieves immediate data from the device.

```
HRESULT GetDeviceState(
  DWORD cbData,
  LPVOID lpvData
);
```

Parameters

cbData
Size of the buffer in the *lpvData* parameter, in bytes.

lpvData
Address of a structure that receives the current state of the device. The format of the data is established by a prior call to the **IDirectInputDevice8::SetDataFormat** method.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
DIERR_NOTINITIALIZED
E_PENDING

Remarks

Before device data can be obtained, set the cooperative level by using the **IDirectInputDevice8::SetCooperativeLevel** method, then set the data format by using **IDirectInputDevice8::SetDataFormat**, and acquire the device by using the **IDirectInputDevice8::Acquire** method.

The five predefined data formats require corresponding device state structures according to the following table:

Data format	State structure
c_dfDIMouse	**DIMOUSESTATE**
c_dfDIMouse2	**DIMOUSESTATE2**
c_dfDIKeyboard	array of 256 bytes
c_dfDIJoystick	**DIJOYSTATE**
c_dfDIJoystick2	**DIJOYSTATE2**

For example, if you passed the *c_dfDIMouse* format to the **IDirectInputDevice8::SetDataFormat** method, you must pass a **DIMOUSESTATE** structure to the **IDirectInputDevice8::GetDeviceState** method.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for

Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::Poll, Polling and Event Notification, Buffered and Immediate Data

IDirectInputDevice8::GetEffectInfo

Obtains information about an effect.

```
HRESULT GetEffectInfo(
  LPDIEFFECTINFO pdei,
  REFGUID rguid
);
```

Parameters

pdei
DIEFFECTINFO structure that receives information about the effect. The caller must initialize the **dwSize** member of the structure before calling this method.

rguid
Reference to (C++) or address of (C) the GUID identifying the effect for which information is being requested.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_DEVICENOTREG
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED
E_POINTER

Note

If this method is called on an non-Force Feedback device, E_POINTER will be returned.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetForceFeedbackState

Retrieves the state of the device's force-feedback system.

```
HRESULT GetForceFeedbackState(
  LPDWORD pdwOut
);
```

Parameters

pdwOut

Location for flags that describe the current state of the device's force-feedback system.

The value is a combination of the following constants:

DIGFFS_ACTUATORSOFF
The device's force-feedback actuators are disabled.

DIGFFS_ACTUATORSON
The device's force-feedback actuators are enabled.

DIGFFS_DEVICELOST
The device suffered an unexpected failure and is in an indeterminate state. It must be reset either by unacquiring and reacquiring the device, or by sending a DISFFC_RESET command.

DIGFFS_EMPTY
The device has no downloaded effects.

DIGFFS_PAUSED
Playback of all active effects has been paused.

DIGFFS_POWEROFF
The force-feedback system is not currently available. If the device cannot report the power state, neither DIGFFS_POWERON nor DIGFFS_POWEROFF is returned.

DIGFFS_POWERON
Power to the force-feedback system is currently available. If the device cannot report the power state, neither DIGFFS_POWERON nor DIGFFS_POWEROFF is returned.

DIGFFS_SAFETYSWITCHOFF
The safety switch is currently off; that is, the device cannot operate. If the device cannot report the state of the safety switch, neither DIGFFS_SAFETYSWITCHON nor DIGFFS_SAFETYSWITCHOFF is returned.

DIGFFS_SAFETYSWITCHON
The safety switch is currently on; that is, the device can operate. If the device cannot report the state of the safety switch, neither

DIGFFS_SAFETYSWITCHON nor DIGFFS_SAFETYSWITCHOFF is returned.

DIGFFS_STOPPED
No effects are playing, and the device is not paused.

DIGFFS_USERFFSWITCHOFF
The user force-feedback switch is currently off; that is, the device cannot operate. If the device cannot report the state of the user force-feedback switch, neither DIGFFS_USERFFSWITCHON nor DIGFFS_USERFFSWITCHOFF is returned.

DIGFFS_USERFFSWITCHON
The user force-feedback switch is currently on; that is, the device can operate. If the device cannot report the state of the user force-feedback switch, neither DIGFFS_USERFFSWITCHON nor DIGFFS_USERFFSWITCHOFF is returned.

Future versions of Microsoft® DirectInput® can define additional flags. Applications should ignore any flags that are not currently defined.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST

DIERR_INVALIDPARAM

DIERR_NOTEXCLUSIVEACQUIRED

DIERR_NOTINITIALIZED

DIERR_UNSUPPORTED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetImageInfo

Retrieves information about a device image for use in a configuration property sheet.

HRESULT GetImageInfo(

LPDIDEVICEIMAGEINFOHEADER *lpdiDevImageInfoHeader*
);

Parameters

lpdiDevImageInfoHeader
Address of a **DIDEVICEIMAGEINFOHEADER** structure that receives information about the device image.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values.

DIERR_INVALIDPARAM
DIERR_MAPFILEFAIL
DIERR_MOREDATA
DIERR_NOTINITIALIZED
DIERR_OBJECTNOTFOUND

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetObjectInfo

Retrieves information about a device object, such as a button or axis.

HRESULT GetObjectInfo(
LPDIDEVICEOBJECTINSTANCE *pdidoi*,
DWORD *dwObj*,
DWORD *dwHow*
);

Parameters

pdidoi
Address of a **DIDEVICEOBJECTINSTANCE** structure to be filled with information about the object. The structure's **dwSize** member must be initialized before this method is called.

dwObj

Value that identifies the object whose information is to be retrieved. The value set for this parameter depends on the value specified in the *dwHow* parameter.

dwHow

Value that specifies how the *dwObj* parameter should be interpreted. This value can be one of the following:

Value	Meaning
DIPH_BYOFFSET	The *dwObj* parameter is the offset into the current data format of the object whose information is being accessed.
DIPH_BYID	The *dwObj* parameter is the object type/instance identifier. This identifier is returned in the **dwType** member of the **DIDEVICEOBJECTINSTANCE** structure returned from a previous call to the **IDirectInputDevice8::EnumObjects** method.
DIPH_BYUSAGE	The *dwObj* parameter contains the HID Usage Page and Usage values of the object, combined by the **DIMAKEUSAGEDWORD** macro.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM

DIERR_NOTINITIALIZED

DIERR_OBJECTNOTFOUND

E_POINTER

Remarks

For compatibility with Microsoft® DirectX® 3, it is also valid to pass a **DIDEVICEOBJECTINSTANCE_DX3** structure with the **dwSize** member initialized to **sizeof(DIDEVICEOBJECTINSTANCE_DX3)**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::GetProperty

Retrieves information about the input device.

```
HRESULT GetProperty(
  REFGUID rguidProp,
  LPDIPROPHEADER pdiph
);
```

Parameters

rguidProp

Reference to (C++) or address of (C) the GUID identifying the property to be retrieved. This can be one of the predefined values or a pointer to a GUID that identifies the property. The following properties are predefined for an input device:

DIPROP_APPDATA

Retrieves the application-defined value associated with an in-game action, as a **DIPROPPOINTER**.

DIPROP_AUTOCENTER

Determines whether device objects are self-centering. The value returned can be DIPROPAUTOCENTER_OFF or DIPROPAUTOCENTER_ON. See **IDirectInputDevice8::SetProperty** for more information.

DIPROP_AXISMODE

Retrieves the axis mode. The retrieved value can be DIPROPAXISMODE_ABS or DIPROPAXISMODE_REL.

DIPROP_BUFFERSIZE

Retrieves the input buffer size.

The buffer size determines the amount of data that the buffer can hold between calls to the **IDirectInputDevice8::GetDeviceData** method before data is lost. This value can be set to 0 to indicate that the application will not be reading buffered data from the device. If the buffer size in the **dwData** member of the **DIPROPDWORD** structure is too large to be supported by the device, the largest possible buffer size is set. To determine whether the requested buffer size was set, retrieve the buffer-size property, and compare the result with the value that you previously attempted to set.

DIPROP_CALIBRATIONMODE

For use by device drivers; not used by applications.

DIPROP_CPOINTS

Retrieves calibration points used for the adjustment of incoming raw data as a **DIPROPCPOINTS** structure.

DIPROP_DEADZONE

Retrieves a value for the dead zone of a joystick, in the range from 0 through 10,000, where 0 indicates that there is no dead zone, 5,000 indicates that the dead zone extends over 50 percent of the physical range of the axis on both sides of center, and 10,000 indicates that the entire physical range of the axis is dead. When the axis is within the dead zone, it is reported as being at the center of its range.

DIPROP_FFGAIN

Retrieves the gain of the device. See **IDirectInputDevice8::SetProperty** for more information.

DIPROP_FFLOAD
Retrieves the memory load for the device. This setting applies to the entire device, rather than to any particular object, so the **dwHow** member of the associated **DIPROPDWORD** structure must be DIPH_DEVICE. The retrieved value is in the range from 0 through 100, indicating the percentage of device memory in use. The device must be acquired in exclusive mode. If it is not, the method will fail with a return value of DIERR_NOTEXCLUSIVEACQUIRED.

DIPROP_GETPORTDISPLAYNAME
Retrieves the human-readable display name of the port to which this device is connected. Not generally used by applications. If no friendly name is returned, the method returns S_FALSE.

DIPROP_GRANULARITY
Retrieves the input granularity. Granularity represents the smallest distance over which the object reports movement. Most axis objects have a granularity of one; that is, all values are possible. Some axes have a larger granularity. For example, the wheel axis on a mouse can have a granularity of 20; that is, all reported changes in position are multiples of 20. In other words, when the user turns the wheel slowly, the device reports a position of 0, then 20, then 40, and so on. This is a read-only property.

DIPROP_GUIDANDPATH
Retrieves the class GUID and device interface (path) for the device. This property lets advanced applications perform operations on a HID that are not supported by Microsoft® DirectInput®. For more information, see the reference for the **DIPROPGUIDANDPATH** structure.

DIPROP_INSTANCENAME
Retrieves the friendly instance name of the device. For more information, see **IDirectInputDevice8::SetProperty**.

DIPROP_JOYSTICKID
Retrieves the instance number of a joystick. This property is not implemented for the mouse or keyboard.

DIPROP_KEYNAME
Retrieves the localized key name for a keyboard key, as a **DIPROPSTRING**.

DIPROP_LOGICALRANGE
Retrieves the range of the raw data returned for axes on a HID. Devices can return negative values.

DIPROP_PHYSICALRANGE
Retrieves the range of data for axes as suggested by the manufacturer of a HID. Values can be negative. Normally DirectInput® returns values from 0 through 0xFFFF, but the range can be made to conform to the manufacturer's suggested range by using DIPROP_RANGE.

DIPROP_PRODUCTNAME

Retrieves the friendly product name of the device. For more information, see **IDirectInputDevice8::SetProperty**.

DIPROP_RANGE
Retrieves the range of values an object can possibly report. For some devices, this is a read-only property.

DIPROP_SATURATION
Retrieves a value for the saturation zones of a joystick, in the range from 0 through 10,000. The saturation level is the point at which the axis is considered to be at its most extreme position. For example, if the saturation level is set to 9,500, the axis reaches the extreme of its range when it has moved 95 percent of the physical distance from its center position (or from the dead zone).

DIPROP_SCANCODE
Retrieves the scan code for a keyboard key, as a **DIPROPDWORD**.

DIPROP_USERNAME
Retrieves the user name for a user currently assigned to a device, as a DIPROPSTRING. User names are set by calling **IDirectInputDevice8::SetActionMap**. If no user name is set, the method returns S_FALSE.

pdiph
Address of the **DIPROPHEADER** portion of a larger property-dependent structure that contains the **DIPROPHEADER** structure as a member. The following structures are used for properties:

DIPROPDWORD
Used for properties represented by a single numerical value.

DIPROPGUIDANDPATH
Used for DIPROP_GUIDANDPATH.

DIPROPRANGE
Used for properties represented by a pair of numerical values. Currently, the only such property is DIPROP_RANGE.

DIPROPSTRING
Used for string properties.

Return Values

If the method succeeds, the return value is DI_OK or S_FALSE.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM

DIERR_NOTEXCLUSIVEACQUIRED

DIERR_NOTINITIALIZED

DIERR_OBJECTNOTFOUND

DIERR_UNSUPPORTED

Remarks

The following C example shows how to obtain the value of the DIPROP_BUFFERSIZE property:

```
DIPROPDWORD dipdw;  // DIPROPDWORD contains a DIPROPHEADER structure.
HRESULT hr;
dipdw.diph.dwSize       = sizeof(DIPROPDWORD);
dipdw.diph.dwHeaderSize = sizeof(DIPROPHEADER);
dipdw.diph.dwObj        = 0; // device property
dipdw.diph.dwHow        = DIPH_DEVICE;

hr = IDirectInputDevice8_GetProperty(pdid, DIPROP_BUFFERSIZE, &dipdw.diph);
if (SUCCEEDED(hr)) {
   // The dipdw.dwData member contains the buffer size.
}
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::SetProperty

IDirectInputDevice8::Initialize

Initializes a DirectInputDevice object. The **IDirectInput8::CreateDevice** method automatically initializes a device after creating it; applications normally do not need to call this method.

```
HRESULT Initialize(
  HINSTANCE hinst,
  DWORD dwVersion,
  REFGUID rguid
);
```

Parameters

hinst

Instance handle to the application or DLL that is creating the DirectInputDevice object. Microsoft® DirectInput® uses this value to determine whether the application or DLL has been certified and to establish any special behaviors that might be necessary for backward compatibility.

It is an error for a DLL to pass the handle to the parent application. For example, a Microsoft ActiveX® control embedded in a Web page that uses DirectInput must pass its own instance handle, and not the handle to the Web browser. This ensures that DirectInput recognizes the control and can enable any special behaviors that may be necessary.

dwVersion

Version number of DirectInput for which the application is designed. This value is normally DIRECTINPUT_VERSION. Passing the version number of a previous version causes DirectInput to emulate that version.

rguid

Reference to (C++) or address of (C) the GUID identifying the instance of the device with which the interface should be associated. The **IDirectInput8::EnumDevices** method can be used to determine which instance GUIDs are supported by the system.

Return Values

If the method succeeds, the return value is DI_OK or S_FALSE.

If the method fails, the return value can be one of the following error values:

DIERR_ACQUIRED
DIERR_DEVICENOTREG

If the method returns S_FALSE, the device had already been initialized with the instance GUID passed in though *rGUID*.

Remarks

If this method fails, the underlying object should be considered to be in an indeterminate state and must be reinitialized before use.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::Poll

Retrieves data from polled objects on a Microsoft® DirectInput® device. If the device does not require polling, calling this method has no effect. If a device that requires polling is not polled periodically, no new data is received from the device. Calling this method causes DirectInput to update the device state, generate input events (if buffered data is enabled), and set notification events (if notification is enabled).

HRESULT Poll()

Parameters

None.

Return Values

If the method succeeds, the return value is DI_OK, or DI_NOEFFECT if the device does not require polling.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST
DIERR_NOTACQUIRED
DIERR_NOTINITIALIZED

Remarks

Before a device data can be polled, the data format must be set by using the **IDirectInputDevice8::SetDataFormat** or **IDirectInputDevice8::SetActionMap** method, and the device must be acquired by using the **IDirectInputDevice8::Acquire** method.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

Polling and Event Notification

IDirectInputDevice8::RunControlPanel

Runs the Microsoft® DirectInput® control panel associated with this device. If the device does not have a control panel associated with it, the default device control panel is launched.

```
HRESULT RunControlPanel(
  HWND hwndOwner,
  DWORD dwFlags
);
```

Parameters

hwndOwner
Parent window handle. If this parameter is NULL, no parent window is used.

dwFlags
Not currently used. Zero is the only valid value.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::SendDeviceData

Sends data to a device that accepts output. The device must be in an acquired state.

```
HRESULT SendDeviceData(
  DWORD cbObjectData,
  LPCDIDEVICEOBJECTDATA rgdod,
  LPDWORD pdwInOut,
  DWORD fl
);
```

Parameters

cbObjectData
Size, in bytes, of a single **DIDEVICEOBJECTDATA** structure.

rgdod
Array of **DIDEVICEOBJECTDATA** structures containing the data to send to the device. It must consist of **pdwInOut* elements.

The **dwOfs** field of each **DIDEVICEOBJECTDATA** structure must contain the device object identifier (as obtained from the **dwType** field of the **DIDEVICEOBJECTINSTANCE** structure) for the device object to which the data is directed. The **dwTimeStamp** and **dwSequence** members must be 0.

pdwInOut

On entry, the variable pointed to by this parameter contains the number of elements in the array pointed to by *rgdod*. On exit, it contains the number of elements sent to the device.

fl

Flags controlling the manner in which data is sent. This can be 0 or the following value:

DISDD_CONTINUE
The device data sent is overlaid on the previously sent device data. See Remarks.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST
DIERR_NOTACQUIRED
DIERR_REPORTFULL
DIERR_UNPLUGGED

Remarks

There is no guarantee that the individual data elements will be sent in a particular order. However, data sent by successive calls to **IDirectInputDevice8::SendDeviceData** is not interleaved. Furthermore, if multiple pieces of data are sent to the same object with a single call, it is unspecified which piece of data is sent.

Consider, for example, a device that can be sent data in packets, each packet describing two pieces of information; call them A and B. Suppose the application attempts to send three data elements: B = 2, A = 1, and B = 0.

The actual device is sent a single packet. The A field of the packet contains the value 1, and the B field of the packet is either 2 or 0.

If the data must to be sent to the device exactly as specified, three calls to **IDirectInputDevice8::SendDeviceData** should be performed, each call sending one data element.

In response to the first call, the device is sent a packet in which the A field is blank and the B field contains the value 2.

In response to the second call, the device is sent a packet in which the A field contains the value 1, and the B field is blank.

Finally, in response to the third call, the device is sent a packet in which the A field is blank and the B field contains the value 0.

If the DISDD_CONTINUE flag is set, the device data sent is overlaid on the previously sent device data. Otherwise, the device data sent does not include the previously sent device data.

For example, suppose a device supports two button outputs, Button0 and Button1. If an application first calls **IDirectInputDevice8::SendDeviceData** passing "Button0 pressed", a packet of the form "Button0 pressed, Button1 not pressed" is sent to the device. If the application then makes another call, passing "Button1 pressed" and the DISDD_CONTINUE flag, a packet of the form "Button0 pressed, Button1 pressed" is sent to the device. However, if the application had not passed the DISDD_CONTINUE flag, the packet sent to the device would have been "Button0 not pressed, Button1 pressed".

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::SendForceFeedbackCommand

Sends a command to the device's force-feedback system.

```
HRESULT SendForceFeedbackCommand(
  DWORD dwFlags
);
```

Parameters

dwFlags
Single value indicating the desired change in state. The value can be one of the following:

DISFFC_CONTINUE
Paused playback of all active effects is to be continued. It is an error to send this command when the device is not in a paused state.

DISFFC_PAUSE
Playback of all active effects is to be paused. This command also stops the clock-on effects so that they continue playing to their full duration when restarted.

While the device is paused, new effects cannot be started, and existing ones cannot be modified. Doing so can cause the subsequent DISFFC_CONTINUE command to fail to perform properly.

To abandon a pause and stop all effects, use the DISFFC_STOPALL or DISFCC_RESET commands.

DISFFC_RESET
The device's force-feedback system is to be put in its startup state. All effects are removed from the device, are no longer valid, and must be recreated if they are to be used again. The device's actuators are disabled.

DISFFC_SETACTUATORSOFF
The device's force-feedback actuators are to be disabled. While the actuators are off, effects continue to play but are ignored by the device. Using the analogy of a sound playback device, they are muted, rather than paused.

DISFFC_SETACTUATORSON
The device's force-feedback actuators are to be enabled.

DISFFC_STOPALL
Playback of any active effects is to be stopped. All active effects are reset, but are still being maintained by the device and are still valid. If the device is in a paused state, that state is lost.

This command is equivalent to calling the **IDirectInputEffect::Stop** method for each effect playing.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED
DIERR_UNSUPPORTED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::SetActionMap

Sets the data format for a device and maps application-defined actions to device objects. It also sets the buffer size for buffered data.

Note

The device must be unacquired prior to calling this method.

```
HRESULT SetActionMap(
  LPCDIACTIONFORMAT lpdiActionFormat,
  LPCWSTR lptszUserName,
  DWORD dwFlags
);
```

Parameters

lpdiActionFormat

Address of a **DIACTIONFORMAT** structure containing information about the action map to be applied.

lptszUserName

Unicode string that specifies the name of the user for which the action map is being set. A value of NULL specifies the user currently logged into the system.

dwFlags

DWORD value that specifies how the action map is applied. This can be one of the following values.

DIDSAM_DEFAULT

Set the action map for this user. If the map differs from the current map, the new settings are saved to disk.

DIDSAM_FORCESAVE

Always save the configuration to disk.

DIDSAM_NOUSER

Reset user ownership for this device in the default configuration property sheet. Resetting user ownership does not remove the current action map.

Return Values

If the method succeeds, the return value can be one of the following error values:

DI_OK

DI_SETTINGSNOTSAVED

DI_WRITEPROTECT

If the method fails, the return value can be one of the following error values.

DIERR_ACQUIRED

DIERR_INVALIDPARAM

Remarks

This method provides the mechanism to change action-to-control mapping from the device defaults. An application must use this method to map its in-game actions to virtual controls.

The user name passed to this method binds a set of action mappings for a device to a specific user. Settings are automatically saved to disk when they differ from the currently applied map. Applications that accept input from multiple users should be very careful when applying action maps to the system mouse or keyboard, as the action maps for each user may conflict.

The method can be called only when the device is not acquired.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::BuildActionMap

IDirectInputDevice8::SetCooperativeLevel

Establishes the cooperative level for this instance of the device. The cooperative level determines how this instance of the device interacts with other instances of the device and the rest of the system.

```
HRESULT SetCooperativeLevel(
  HWND hwnd,
  DWORD dwFlags
);
```

Parameters

hwnd
Window handle to be associated with the device. This parameter must be a valid top-level window handle that belongs to the process. The window associated with the device must not be destroyed while it is still active in a Microsoft® DirectInput® device.

dwFlags
Flags that describe the cooperative level associated with the device. The following flags are defined:

DISCL_BACKGROUND
The application requires background access. If background access is granted, the device can be acquired at any time, even when the associated window is not the active window.

DISCL_EXCLUSIVE
The application requires exclusive access. If exclusive access is granted, no other instance of the device can obtain exclusive access to the device while it is acquired. However, nonexclusive access to the device is always permitted, even if another application has obtained exclusive access.

An application that acquires the mouse or keyboard device in exclusive mode should always unacquire the devices when it receives WM_ENTERSIZEMOVE and WM_ENTERMENULOOP messages. Otherwise, the user cannot manipulate the menu or move and resize the window.

DISCL_FOREGROUND
The application requires foreground access. If foreground access is granted, the device is automatically unacquired when the associated window moves to the background.

DISCL_NONEXCLUSIVE
The application requires nonexclusive access. Access to the device does not interfere with other applications that are accessing the same device.

DISCL_NOWINKEY
Disable the Windows logo key. Setting this flag ensures that the user cannot inadvertently break out of the application. Note, however, that DISCL_NOWINKEY has no effect when the default action mapping UI is displayed, and the Microsoft Windows® logo key will operate normally as long as that UI is present.

Applications must specify either DISCL_FOREGROUND or DISCL_BACKGROUND; it is an error to specify both or neither. Similarly, applications must specify either DISCL_EXCLUSIVE or DISCL_NONEXCLUSIVE.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM

DIERR_NOTINITIALIZED

E_HANDLE

Remarks

If the system mouse is acquired in exclusive mode, the pointer is removed from the screen until the device is unacquired. This applies only to a mouse created by passing GUID_SysMouse to **IDirectInput8::CreateDevice**.

Applications must call this method before acquiring the device by using the **IDirectInputDevice8::Acquire** method.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

Cooperative Levels

IDirectInputDevice8::SetDataFormat

Sets the data format for the Microsoft® DirectInput® device.

```
HRESULT SetDataFormat(
  LPCDIDATAFORMAT lpdf
);
```

Parameters

lpdf

Address of a structure that describes the format of the data that the DirectInputDevice should return. An application can define its own **DIDATAFORMAT** structure or use one of the following predefined global variables:

- *c_dfDIKeyboard*
- *c_dfDIMouse*
- *c_dfDIMouse2*
- *c_dfDIJoystick*
- *c_dfDIJoystick2*

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_ACQUIRED
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

The data format must be set before the device can be acquired by using the **IDirectInputDevice8::Acquire** method. It is necessary to set the data format only once. The data format cannot be changed while the device is acquired.

If the application is using action mapping, the data format is set instead by the call to **IDirectInputDevice8::SetActionMap**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::GetDeviceState

IDirectInputDevice8::SetEventNotification

Specifies an event that is to be set when the device state changes. It is also used to turn off event notification.

```
HRESULT SetEventNotification(
  HANDLE hEvent
);
```

Parameters

hEvent

Handle to the event that is to be set when the device state changes. Microsoft® DirectInput® uses the Microsoft® Win32® **SetEvent** function on the handle when the state of the device changes. If the *hEvent* parameter is NULL, notification is disabled.

The application can create the handle as either a manual-reset or autoreset event by using the Microsoft® Win32® **CreateEvent** function. If the event is created as an autoreset event, the operating system automatically resets the event once a wait has been satisfied. If the event is created as a manual-reset event, it is the application's responsibility to call the Win32 **ResetEvent** function to reset it.

DirectInput does not call the Win32 **ResetEvent** function for event notification handles. Most applications create the event as an automatic-reset event.

Return Values

If the method succeeds, the return value is DI_OK or DI_POLLEDDEVICE.

If the method fails, the return value can be one of the following error values:

DIERR_ACQUIRED
DIERR_HANDLEEXISTS
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

A device state change is defined as any of the following:

- A change in the position of an axis
- A change in the state (pressed or released) of a button
- A change in the direction of a POV control
- Loss of acquisition

Do not call the Win32 **CloseHandle** function on the event while it has been selected into a DirectInputDevice object. You must call this method with the *hEvent* parameter set to NULL before closing the event handle.

The event notification handle cannot be changed while the device is acquired. If the function is successful, the application can use the event handle like any other Win32 event handle.

The following code example checks whether the handle is currently set without blocking:

```
dwResult = WaitForSingleObject(hEvent, 0);
if (dwResult == WAIT_OBJECT_0) {
    // Event is set. If the event was created as
    // autoreset, it has also been reset.
}
```

The following code example illustrates blocking indefinitely until the event is set. This behavior is strongly discouraged because the thread does not respond to the system until the wait is satisfied. In particular, the thread does not respond to Windows® messages.

```
dwResult = WaitForSingleObject(hEvent, INFINITE);
if (dwResult == WAIT_OBJECT_0) {
    // Event has been set. If the event was created
```

```
    // as autoreset, it has also been reset.
}
```

The following code example illustrates a typical message loop for a message-based application that uses two events:.

```
HANDLE ah[2] = { hEvent1, hEvent2 };

while (TRUE) {

    dwResult = MsgWaitForMultipleObjects(2, ah, FALSE,
                    INFINITE, QS_ALLINPUT);
    switch (dwResult) {
    case WAIT_OBJECT_0:
        // Event 1 has been set. If the event was created as
        // autoreset, it has also been reset.
        ProcessInputEvent1();
        break;

    case WAIT_OBJECT_0 + 1:
        // Event 2 has been set. If the event was created as
        // autoreset, it has also been reset.
        ProcessInputEvent2();
        break;

    case WAIT_OBJECT_0 + 2:
        // A Windows message has arrived. Process
        // messages until there aren't any more.
        while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)){
            if (msg.message == WM_QUIT) {
                goto exitapp;
            }
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        break;

    default:
        // Unexpected error.
        Panic();
        break;
    }
}
```

The following code example illustrates a typical application loop for a non-message-based application that uses two events:

```
HANDLE ah[2] = { hEvent1, hEvent2 };
```

```
DWORD dwWait = 0;

while (TRUE) {

   dwResult = MsgWaitForMultipleObjects(2, ah, FALSE,
                                dwWait, QS_ALLINPUT);
   dwWait = 0;

   switch (dwResult) {
   case WAIT_OBJECT_0:
      // Event 1 has been set. If the event was
      // created as autoreset, it has also
      // been reset.
      ProcessInputEvent1();
      break;

   case WAIT_OBJECT_0 + 1:
      // Event 2 has been set. If the event was
      // created as autoreset, it has also
      // been reset.
      ProcessInputEvent2();
      break;

   case WAIT_OBJECT_0 + 2:
      // A Windows message has arrived. Process
      // messages until there aren't any more.
      while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)){
         if (msg.message == WM_QUIT) {
            goto exitapp;
         }
         TranslateMessage(&msg);
         DispatchMessage(&msg);
      }
      break;

   default:
      // No input or messages waiting.
      // Do a frame of the game.
      // If the game is idle, tell the next wait
      // to wait indefinitely for input or a message.
      if (!DoGame()) {
         dwWait = INFINITE;
      }
      break;
   }
}
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

Polling and Event Notification

IDirectInputDevice8::SetProperty

Sets properties that define the device behavior. These properties include input buffer size and axis mode.

```
HRESULT SetProperty(
  REFGUID rguidProp,
  LPCDIPROPHEADER pdiph
);
```

Parameters

rguidProp
Reference to (C++) or address of (C) the GUID identifying the property to be set. This can be one of the predefined values, or a pointer to a GUID that identifies the property. The following property values are predefined for an input device:

DIPROP_APPDATA
Sets the application-defined value associated with an in-game action, as a **DIPROPPOINTER**.

DIPROP_AUTOCENTER
Specifies whether device objects are self-centering. This setting applies to the entire device, rather than to any particular object, so the **dwHow** member of the associated **DIPROPDWORD** structure must be DIPH_DEVICE.

The **dwData** member can be one of the following values.

DIPROPAUTOCENTER_OFF: The device should not automatically center when the user releases the device. An application that uses force feedback should disable autocentering before playing effects.

DIPROPAUTOCENTER_ON: The device should automatically center when the user releases the device.

Not all devices support the autocenter property.

DIPROP_AXISMODE
Sets the axis mode. The value being set (DIPROPAXISMODE_ABS or DIPROPAXISMODE_REL) must be specified in the **dwData** member of the associated **DIPROPDWORD** structure. See the description of the *pdiph* parameter for more information.

This setting applies to the entire device, so the **dwHow** member of the associated **DIPROPDWORD** structure must be set to DIPH_DEVICE.

DIPROP_BUFFERSIZE

Sets the input buffer size. The value being set must be specified in the **dwData** member of the associated **DIPROPDWORD** structure. See Remarks.

This setting applies to the entire device, so the **dwHow** member of the associated **DIPROPDWORD** structure must be set to DIPH_DEVICE.

DIPROP_CALIBRATIONMODE

Enables the application to specify whether Microsoft® DirectInput® should retrieve calibrated or uncalibrated data from an axis. By default, DirectInput retrieves calibrated data.

Setting the calibration mode for the entire device is equivalent to setting it for each axis individually.

The **dwData** member of the **DIPROPDWORD** structure can be one of the following values:

DIPROPCALIBRATIONMODE_COOKED: DirectInput should return data after applying calibration information. This is the default mode.

DIPROPCALIBRATIONMODE_RAW: DirectInput should return raw, uncalibrated data. This mode is typically used only by Control Panel–type applications. Note that raw data might include negative values.

Setting a device into raw mode causes the dead zone, saturation, and range settings to be ignored.

DIPROP_CPOINTS

Sets calibration points used for the adjustment of incoming raw data. The values being set must be specified as CPOINT types in the **cp** array of the associated **DIPROPCPOINTS** structure.

This setting applies to individual device objects, so the **dwHow** member of the associated **DIPROPHEADER** structure must be set to either DIPH_BYID or DIPH_BYOFFSET.

DIPROP_DEADZONE

Sets the value for the dead zone of a joystick, in the range from 0 through 10,000, where 0 indicates that there is no dead zone, 5,000 indicates that the dead zone extends over 50 percent of the physical range of the axis on both sides of center, and 10,000 indicates that the entire physical range of the axis is dead. When the axis is within the dead zone, it is reported as being at the center of its range.

This setting can be applied to either the entire device or to a specific axis.

DIPROP_FFGAIN

Sets the gain for the device. This setting applies to the entire device, rather than to any particular object, so the **dwHow** member of the associated **DIPROPDWORD** structure must be DIPH_DEVICE.

The **dwData** member contains a gain value that is applied to all effects created on the device. The value is an integer in the range from 0 through 10,000, specifying the amount by which effect magnitudes should be scaled for the

device. For example, a value of 10,000 indicates that all effect magnitudes are to be taken at face value. A value of 9,000 indicates that all effect magnitudes are to be reduced to 90% of their nominal magnitudes.

Setting a gain value is useful when an application wants to scale down the strength of all force-feedback effects uniformly, based on user preferences.

Unlike other properties, the gain can be set when the device is in an acquired state.

DIPROP_INSTANCENAME

This property exists for advanced applications that want to change the friendly instance name of a device (as returned in the **tszInstanceName** member of the **DIDEVICEINSTANCE** structure) to distinguish it from similar devices that are plugged in simultaneously. Most applications should have no need to change the friendly name.

This setting applies to the entire device, so the **dwHow** member of the associated **DIPROPDWORD** structure must be set to DIPH_DEVICE.

The *pdiph* parameter must be a pointer to the **diph** member of a **DIPROPSTRING** structure.

DIPROP_PRODUCTNAME

This property exists for advanced applications that want to change the friendly product name of a device (as returned in the **tszProductName** member of the **DIDEVICEINSTANCE** structure) to distinguish it from similar devices which are plugged in simultaneously. Most applications should have no need to change the friendly name.

This setting applies to the entire device, so the **dwHow** member of the associated **DIPROPDWORD** structure must be set to DIPH_DEVICE.

The *pdiph* parameter must be a pointer to the **diph** member of a **DIPROPSTRING** structure.

DIPROP_RANGE

Sets the range of values an object can possibly report. The minimum and maximum values are taken from the **lMin** and **lMax** members of the associated **DIPROPRANGE** structure.

For some devices, this is a read-only property.

You cannot set a reverse range; **lMax** must be greater than **lMin**.

DIPROP_SATURATION

Sets the value for the saturation zones of a joystick, in the range from 0 through 10,000. The saturation level is the point at which the axis is considered to be at its most extreme position. For example, if the saturation level is set to 9,500, the axis reaches the extreme of its range when it has moved 95 percent of the physical distance from its center position (or from the dead zone).

This setting can be applied to either the entire device or a specific axis.

pdiph

Address of the **DIPROPHEADER** structure contained within the type-specific property structure.

Return Values

If the method succeeds, the return value is DI_OK or DI_PROPNOEFFECT.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED
DIERR_OBJECTNOTFOUND
DIERR_UNSUPPORTED

Remarks

The buffer size determines the amount of data that the buffer can hold between calls to the **IDirectInputDevice8::GetDeviceData** method before data is lost. This value may be set to 0 to indicate that the application does not read buffered data from the device. If the buffer size in the **dwData** member of the **DIPROPDWORD** structure is too large to be supported by the device, the largest possible buffer size is set. To determine whether the requested buffer size was set, retrieve the buffer-size property and compare the result with the value that you previously attempted to set.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::GetProperty

IDirectInputDevice8::Unacquire

Releases access to the device.

```
HRESULT Unacquire();
```

Parameters

None.

Return Values

The return value is DI_OK is the device was unacquired, or DI_NOEFFECT if the device was not in an acquired state.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputDevice8::WriteEffectToFile

Saves information about one or more force-feedback effects to a file that can be read by using **IDirectInputDevice8::EnumEffectsInFile**. This method is chiefly of interest to those wanting to write their own force-authoring applications.

```
HRESULT WriteEffectToFile(
  LPCSTR lpszFileName,
  DWORD dwEntries,
  LPCDIFILEEFFECT rgDiFileEft,
  DWORD dwFlags
);
```

Parameters

lpszFileName
Name of the RIFF file.

dwEntries
Number of structures in the *rgDiFileEft* array.

rgDiFileEft
Array of **DIFILEEFFECT** structures.

dwFlags
Flags which control how the effect should be written. This can be DIFEF_DEFAULT (= 0) or the following value:

DIFEF_INCLUDENONSTANDARD
Includes effects that are not defined by Microsoft® DirectInput®. If this flag is not specified, only effects with GUIDs defined in Dinput.h, such as GUID_ConstantForce, are written.

Return Values

If the method succeeds, it returns DI_OK.

If it fails, the return value can be DIERR_INVALIDPARAM.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for

Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::EnumEffectsInFile

IDirectInputEffect

Applications use the methods of the **IDirectInputEffect** interface to manage effects of force-feedback devices.

The interface is obtained by using the **IDirectInputDevice8::CreateEffect** method.

The methods of the **IDirectInputEffect** interface can be organized into the following groups.

Effect information	**GetEffectGuid**
	GetEffectStatus
	GetParameters
Effect manipulation	**Download**
	Initialize
	SetParameters
	Start
	Stop
	Unload
Miscellaneous	**Escape**

The **IDirectInputEffect** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

IUnknown	**AddRef**
	QueryInterface
	Release

The **LPDIRECTINPUTEFFECT** type is defined as a pointer to the **IDirectInputEffect** interface:

```
typedef struct IDirectInputEffect    *LPDIRECTINPUTEFFECT;
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::Download

Places the effect on the device. If the effect is already on the device, the existing effect is updated to match the values set by the **IDirectInputEffect::SetParameters** method.

HRESULT Download(void);

Parameters

None.

Return Values

If the method succeeds, the return value is DI_OK or S_FALSE.

If the method fails, the return value can be one of the following error values:

DIERR_DEVICEFULL
DIERR_EFFECTPLAYING
DIERR_INCOMPLETEEFFECT
DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED

If the method returns S_FALSE, the effect has already been downloaded to the device.

Remarks

It is valid to update an effect while it is playing. The semantics of such an operation are explained in the reference for **IDirectInputEffect::SetParameters**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::Escape

Sends a hardware-specific command to the driver.

HRESULT Escape(
LPDIEFFESCAPE *pesc*

```
);
```

Parameters

pesc

DIEFFESCAPE structure that describes the command to be sent. On success, the **cbOutBuffer** member contains the number of bytes of the output buffer used.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_NOTINITIALIZED
DIERR_DEVICEFULL

Other device-specific error codes are also possible. Ask the hardware manufacturer for details.

Remarks

Because each driver implements different escapes, it is the application's responsibility to ensure that it is sending the escape to the correct driver by comparing the value of the **guidFFDriver** member of the **DIDEVICEINSTANCE** structure against the value the application is expecting.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::GetEffectGuid

Retrieves the GUID for the effect represented by the **IDirectInputEffect** object.

```
HRESULT GetEffectGuid(
  LPGUID pguid
);
```

Parameters

pguid

GUID structure that is filled by the method.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

Additional information about the effect can be obtained by passing the GUID to **IDirectInputDevice8::GetEffectInfo**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::GetEffectStatus

Retrieves the status of an effect.

```
HRESULT GetEffectStatus(
  LPDWORD pdwFlags
);
```

Parameters

pdwFlags
Status flags for the effect. The value can be 0 or one or more of the following constants:

DIEGES_PLAYING
The effect is playing.

DIEGES_EMULATED
The effect is emulated.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::GetParameters

Retrieves information about an effect.

```
HRESULT GetParameters(
  LPDIEFFECT peff,
  DWORD dwFlags
);
```

Parameters

peff
Address of a **DIEFFECT** structure that receives effect information. The **dwSize** member must be filled in by the application before calling this method.

dwFlags
Flags that specify which parts of the effect information are to be retrieved. The value can be 0 or one or more of the following constants:

DIEP_ALLPARAMS
The union of all other DIEP_* flags, indicating that all members of the **DIEFFECT** structure are being requested.

DIEP_ALLPARAMS_DX5
The union of all other DIEP_* flags except the DIEP_STARTDELAY flag.

DIEP_AXES
The **cAxes** and **rgdwAxes** members should receive data. The **cAxes** member on entry contains the size (in **DWORD**s) of the buffer pointed to by the **rgdwAxes** member. If the buffer is too small, the method returns DIERR_MOREDATA and sets **cAxes** to the necessary size of the buffer.

DIEP_DIRECTION
The **cAxes** and **rglDirection** members should receive data. The **cAxes** member on entry contains the size (in **DWORD**s) of the buffer pointed to by the **rglDirection** member. If the buffer is too small, the **GetParameters** method returns DIERR_MOREDATA and sets **cAxes** to the necessary size of the buffer.

The **dwFlags** member must include at least one of the coordinate system flags (DIEFF_CARTESIAN, DIEFF_POLAR, or DIEFF_SPHERICAL). Microsoft® DirectInput® returns the direction of the effect in one of the coordinate systems you specified, converting between coordinate systems as necessary. On exit, exactly one of the coordinate system flags is set in the **dwFlags** member, indicating which coordinate system DirectInput used. In

particular, passing all three coordinate system flags retrieves the coordinates in exactly the same format in which they were set.

DIEP_DURATION
The **dwDuration** member should receive data.

DIEP_ENVELOPE
The **lpEnvelope** member points to a **DIENVELOPE** structure that should receive data. If the effect does not have an envelope associated with it, the **lpEnvelope** member is set to NULL.

DIEP_GAIN
The **dwGain** member should receive data.

DIEP_SAMPLEPERIOD
The **dwSamplePeriod** member should receive data.

DIEP_STARTDELAY
The **dwStartDelay** member should receive data.

DIEP_TRIGGERBUTTON
The **dwTriggerButton** member should receive data.

DIEP_TRIGGERREPEATINTERVAL
The **dwTriggerRepeatInterval** member should receive data.

DIEP_TYPESPECIFICPARAMS
The **lpvTypeSpecificParams** member points to a buffer whose size is specified by the **cbTypeSpecificParams** member. On return, the buffer is filled in with the type-specific data associated with the effect, and the **cbTypeSpecificParams** member contains the number of bytes copied. If the buffer supplied by the application is too small to contain all the type-specific data, the method returns DIERR_MOREDATA, and the **cbTypeSpecificParams** member contains the required size of the buffer in bytes.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INVALIDPARAM

DIERR_MOREDATA

DIERR_NOTINITIALIZED

Remarks

Common errors resulting in a DIERR_INVALIDPARAM error include not setting the **dwSize** member of the **DIEFFECT** structure, passing invalid flags, or not setting up the members in the **DIEFFECT** structure properly in preparation for receiving the effect information. For example, if information is to be retrieved in the **dwTriggerButton** member, the **dwFlags** member must be set to either

DIEFF_OBJECTIDS or DIEFF_OBJECTOFFSETS so that DirectInput knows how to describe the button.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::Initialize

Initializes a DirectInputEffect object.

```
HRESULT Initialize(
  HINSTANCE hinst,
  DWORD dwVersion,
  REFGUID rguid
);
```

Parameters

hinst
Instance handle to the application or DLL that is creating the DirectInputEffect object. Microsoft® DirectInput® uses this value to determine whether the application or DLL has been certified and to establish any special behaviors that might be necessary for backward compatibility. It is an error for a DLL to pass the handle to the parent application.

dwVersion
Version number of DirectInput for which the application is designed. This value is normally DIRECTINPUT_VERSION. Passing the version number of a previous version causes DirectInput to emulate that version.

rguid
Reference to (C++) or address of (C) the GUID identifying the effect with which the interface is associated. The **IDirectInputDevice8::EnumEffects** method can be used to determine which effect GUIDs are supported by the device.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be DIERR_DEVICENOTREG.

Remarks

If this method fails, the underlying object should be considered to be an indeterminate state and needs to be reinitialized before it can be subsequently used.

The **IDirectInputDevice8::CreateEffect** method automatically initializes the effect after creating it. Applications normally do not need to call the **Initialize** method.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::SetParameters

Sets the characteristics of an effect.

```
HRESULT SetParameters(
  LPCDIEFFECT peff,
  DWORD dwFlags
);
```

Parameters

peff
DIEFFECT structure that contains effect information. The **dwSize** member must be filled in by the application before calling this method, as well as any members specified by corresponding bits in the *dwFlags* parameter.

dwFlags
Flags that specify which portions of the effect information are to be set and how the downloading of the parameters should be handled. The value can be 0 or one or more of the following constants:

DIEP_AXES
The **cAxes** and **rgdwAxes** members contain data.

DIEP_DIRECTION
The **cAxes** and **rglDirection** members contain data. The **dwFlags** member specifies (with DIEFF_CARTESIAN or DIEFF_POLAR) the coordinate system in which the values should be interpreted.

DIEP_DURATION
The **dwDuration** member contains data.

DIEP_ENVELOPE
The **lpEnvelope** member points to a **DIENVELOPE** structure that contains data. To detach any existing envelope from the effect, pass this flag and set the **lpEnvelope** member to NULL.

DIEP_GAIN
The **dwGain** member contains data.

DIEP_NODOWNLOAD

Suppress the automatic **IDirectInputEffect::Download** that is normally performed after the parameters are updated. See Remarks.

DIEP_NORESTART
Suppress the stopping and restarting of the effect to change parameters. See Remarks.

DIEP_SAMPLEPERIOD
The **dwSamplePeriod** member contains data.

DIEP_START
The effect is to be started (or restarted if it is currently playing) after the parameters are updated. By default, the play state of the effect is not altered.

DIEP_STARTDELAY
The **dwStartDelay** member contains data.

DIEP_TRIGGERBUTTON
The **dwTriggerButton** member contains data.

DIEP_TRIGGERREPEATINTERVAL
The **dwTriggerRepeatInterval** member contains data.

DIEP_TYPESPECIFICPARAMS
The **lpvTypeSpecificParams** and **cbTypeSpecificParams** members of the **DIEFFECT** structure contain the address and size of type-specific data for the effect.

Return Values

If the method succeeds, the return value is one of the following:

DI_DOWNLOADSKIPPED
DI_EFFECTRESTARTED
DI_OK
DI_TRUNCATED
DI_TRUNCATEDANDRESTARTED

If the method fails, the return value can be one of the following error values:

DIERR_NOTINITIALIZED
DIERR_INCOMPLETEEFFECT
DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_EFFECTPLAYING

Remarks

The **dwDynamicParams** member of the **DIEFFECTINFO** structure for the effect specifies which parameters can be dynamically updated while the effect is playing.

The **IDirectInputEffect::SetParameters** method automatically downloads the effect, but this behavior can be suppressed by setting the DIEP_NODOWNLOAD flag. If automatic download has been suppressed, you can manually download the effect by invoking the **IDirectInputEffect::Download** method.

If the effect is playing while the parameters are changed, the new parameters take effect as if they were the parameters when the effect started.

For example, suppose a periodic effect with a duration of three seconds is started. After two seconds, the direction of the effect is changed. The effect then continues for one additional second in the new direction. The envelope, phase, amplitude, and other parameters of the effect continue smoothly, as if the direction had not changed.

In the same situation, if after two seconds the duration of the effect were changed to 1.5 seconds, the effect would stop.

Normally, if the driver cannot update the parameters of a playing effect, the driver is permitted to stop the effect, update the parameters, and then restart the effect. Passing the DIEP_NORESTART flag suppresses this behavior. If the driver cannot update the parameters of an effect while it is playing, the error code DIERR_EFFECTPLAYING is returned, and the parameters are not updated.

No more than one of the DIEP_NODOWNLOAD, DIEP_START, and DIEP_NORESTART flags should be set. (It is also valid to pass none of them.)

These three flags control download and playback behavior as follows:

If DIEP_NODOWNLOAD is set, the effect parameters are updated but not downloaded to the device.

If the DIEP_START flag is set, the effect parameters are updated and downloaded to the device, and the effect is started just as if the **IDirectInputEffect::Start** method had been called with the *dwIterations* parameter set to 1 and with no flags. (Combining the update with DIEP_START is slightly faster than calling **Start** separately, because it requires less information to be transmitted to the device.)

If neither DIEP_NODOWNLOAD nor DIEP_START is set and the effect is not playing, the parameters are updated and downloaded to the device.

If neither DIEP_NODOWNLOAD nor DIEP_START is set and the effect is playing, the parameters are updated if the device supports on-the-fly updating. Otherwise the behavior depends on the state of the DIEP_NORESTART flag. If it is set, the error code DIERR_EFFECTPLAYING is returned. If it is clear, the effect is stopped, the parameters are updated, and the effect is restarted.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::Start

Begins playing an effect. If the effect is already playing, it is restarted from the beginning. If the effect has not been downloaded or has been modified since its last download, it is downloaded before being started. This default behavior can be suppressed by passing the DIES_NODOWNLOAD flag.

```
HRESULT Start(
  DWORD dwIterations,
  DWORD dwFlags
);
```

Parameters

dwIterations

Number of times to play the effect in sequence. The envelope is re-articulated with each iteration.

To play the effect exactly once, pass 1. To play the effect repeatedly until explicitly stopped, pass INFINITE. To play the effect until explicitly stopped without re-articulating the envelope, modify the effect parameters with the **IDirectInputEffect::SetParameters** method, and change the **dwDuration** member to INFINITE.

dwFlags

Flags that describe how the effect should be played by the device. The value can be 0 or one or more of the following values:

DIES_SOLO

All other effects on the device should be stopped before the specified effect is played. If this flag is omitted, the effect is mixed with existing effects already started on the device.

DIES_NODOWNLOAD

Do not automatically download the effect.

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INCOMPLETEEFFECT

DIERR_INVALIDPARAM

DIERR_NOTEXCLUSIVEACQUIRED

DIERR_NOTINITIALIZED

DIERR_UNSUPPORTED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

Not all devices support multiple iterations.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::Stop

Stops playing an effect.

```
HRESULT Stop(void);
```

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

IDirectInputEffect::Unload

Removes the effect from the device. If the effect is playing, it is automatically stopped before it is unloaded.

```
HRESULT Unload(void);
```

Return Values

If the method succeeds, the return value is DI_OK.

If the method fails, the return value can be one of the following error values:

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

Callback Functions

The following seven functions are prototype callback functions for use with various enumeration methods. Applications can declare one of these callback functions under any name and define it in any way, but the parameter and return types must be the same as in the prototype.

- **DIConfigureDevicesCallback**
- **DIEnumCreatedEffectObjectsCallback**
- **DIEnumDeviceObjectsCallback**
- **DIEnumDevicesBySemanticsCallback**
- **DIEnumDevicesCallback**
- **DIEnumEffectsCallback**
- **DIEnumEffectsInFileCallback**

DIConfigureDevicesCallback

Application-defined callback function that is called each time the property sheet image is refreshed. This function needs to be supplied only when the application is displaying the property sheet within its own window.

BOOL CALLBACK DIConfigureDevicesCallback(
lUnknown FAR* *lpDDSTarget*,
LPVOID *pvRef*

```
);
```

Parameters

lpDDSTarget
Pointer to the **IUnknown** interface of the surface in the **lpUnkDDSTarget** member of the **DICONFIGUREDEVICESPARAMS** structure that was passed to **IDirectInput8::ConfigureDevices**.

pvRef
Application-defined value passed to **IDirectInput8::ConfigureDevices** as the *pvRef* parameter.

Return Values

Can return TRUE or FALSE. DirectInput® does not use the return value.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: Use Dinput8.lib.

DIEnumCreatedEffectObjectsCallback

Application-defined callback function that receives DirectInputDevice effects as a result of a call to the **IDirectInputDevice8::EnumCreatedEffectObjects** method.

```
BOOL CALLBACK DIEnumCreatedEffectObjectsCallback(
  LPDIRECTINPUTEFFECT peff,
  LPVOID pvRef
);
```

Parameters

peff
Address of an effect object that has been created.

pvRef
The application-defined value passed to **EnumCreatedEffectObjects** as the *pvRef* parameter.

Return Values

Returns DIENUM_CONTINUE to continue the enumeration or DIENUM_STOP to stop the enumeration.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: User-defined.

DIEnumDeviceObjectsCallback

Application-defined callback function that receives DirectInputDevice objects as a result of a call to the **IDirectInputDevice8::EnumObjects** method.

```
BOOL CALLBACK DIEnumDeviceObjectsCallback(
  LPCDIDEVICEOBJECTINSTANCE lpddoi,
  LPVOID pvRef
);
```

Parameters

lpddoi
DIDEVICEOBJECTINSTANCE structure that describes the object being enumerated.

pvRef
The application-defined value passed to **EnumObjects** as the *pvRef* parameter.

Return Values

Returns DIENUM_CONTINUE to continue the enumeration or DIENUM_STOP to stop the enumeration.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: User-defined.

DIEnumDevicesBySemanticsCallback

Application-defined callback function that receives DirectInput® devices as a result of a call to the **IDirectInput8::EnumDevicesBySemantics** method.

```
BOOL CALLBACK DIEnumDevicesBySemanticsCallback(
  LPCDIDEVICEINSTANCE lpddi,
  LPDIRECTINPUTDEVICE8 lpdid,
  DWORD dwFlags,
  DWORD dwRemaining,
  LPVOID pvRef
);
```

Parameters

lpddi

Address of a **DIDEVICEINSTANCE** structure that describes the device instance.

lpdid

Pointer to the **IDirectInputDevice8** interface for the device.

dwFlags

Flags that provide information about why the device is being enumerated. This can be a combination of action-mapping flags and one usage flag. At least one action-mapping flag is always present.

The following action-mapping flags are defined.

DIEDBS_MAPPEDPRI1

The device is being enumerated because priority 1 actions can be mapped to the device.

DIEDBS_MAPPEDPRI2

The device is being enumerated because priority 2 actions can be mapped to the device.

The following usage flags are defined.

DIEDBS_RECENTDEVICE

The device is being enumerated because the commands described by the action-mapping flags were recently used.

DIEDBS_NEWDEVICE

The device is being enumerated because the device was installed recently. Devices described by this flag have not been used with this game before.

dwRemaining

Number of devices, after this one, remaining to be enumerated.

pvRef

The application-defined value passed to **IDirectInput8::EnumDevicesBySemantics** as the *pvRef* parameter.

Return Values

Returns DIENUM_CONTINUE to continue the enumeration or DIENUM_STOP to stop the enumeration.

Remarks

If a single hardware device can function as more than one DirectInput® device type, it is enumerated as each device type that it supports. For example, a keyboard with a built-in mouse is enumerated twice: once as a keyboard and once as a mouse. The product GUID is the same for each device, however.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: User-defined.

See Also

DIEnumDevicesCallback

DIEnumDevicesCallback

Application-defined callback function that receives DirectInput® devices as a result of a call to the **IDirectInput8::EnumDevices** method.

```
BOOL CALLBACK DIEnumDevicesCallback(
  LPCDIDEVICEINSTANCE lpddi,
  LPVOID pvRef
);
```

Parameters

lpddi
Address of a **DIDEVICEINSTANCE** structure that describes the device instance.

pvRef
The application-defined value passed to **EnumDevices** or **EnumDevicesBySemantics** as the *pvRef* parameter.

Return Values

Returns DIENUM_CONTINUE to continue the enumeration or DIENUM_STOP to stop the enumeration.

Remarks

If a single hardware device can function as more than one DirectInput device type, it is enumerated as each device type that it supports. For example, a keyboard with a built-in mouse is enumerated twice: once as a keyboard and once as a mouse. The product GUID is the same for each device, however.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: User-defined.

See Also

DIEnumDevicesBySemanticsCallback

DIEnumEffectsCallback

Application-defined callback function used with the **IDirectInputDevice8::EnumEffects** method.

```
BOOL CALLBACK DIEnumEffectsCallback(
  LPCDIEFFECTINFO pdei,
  LPVOID pvRef
);
```

Parameters

pdei
DIEFFECTINFO structure that describes the enumerated effect.

pvRef
The application-defined value passed to **EnumEffects** as the *pvRef* parameter.

Return Values

Returns DIENUM_CONTINUE to continue the enumeration, or DIENUM_STOP to stop it.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: User-defined.

DIEnumEffectsInFileCallback

Application-defined callback function used with the **IDirectInputDevice8::EnumEffectsInFile** method.

```
BOOL CALLBACK DIEnumEffectsInFileCallback(
  LPCDIFILEEFFECT lpDiFileEf,
  LPVOID pvRef
);
```

Parameters

lpDiFileEf
DIFILEEFFECT structure that describes the enumerated effect.

pvRef
The application-defined value passed to **EnumEffectsInFile** as the *pvRef* parameter.

Return Values

Returns DIENUM_CONTINUE to continue the enumeration, or DIENUM_STOP to stop it.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: User-defined.

Functions

This section is a reference for DirectInput® functions other than COM interface methods and callback functions.

The following function falls into this category:

- **DirectInput8Create**

DirectInput8Create

Creates a DirectInput® object and returns an **IDirectInput8** or later interface.

```
HRESULT WINAPI DirectInput8Create(
  HINSTANCE hinst,
  DWORD dwVersion,
  REFIID riidltf,
  LPVOID* ppvOut,
  LPUNKNOWN punkOuter
);
```

Parameters

hinst

Instance handle to the application or DLL that is creating the DirectInput object. DirectInput uses this value to determine whether the application or DLL has been certified and to establish any special behaviors that might be necessary for backward compatibility.

It is an error for a DLL to pass the handle to the parent application. For example, an ActiveX® control embedded in a Web page that uses DirectInput must pass its own instance handle, and not the handle to the Web browser. This ensures that DirectInput recognizes the control and can enable any special behaviors that might be necessary.

dwVersion

Version number of DirectInput for which the application is designed. This value is normally DIRECTINPUT_VERSION. If the application defines DIRECTINPUT_VERSION before including Dinput.h, the value must be greater than 0x0700. For earlier versions, use **DirectInputCreateEx,** which is in Dinput.lib.

riidltf

Unique identifier of the desired interface. For DirectX 8.0, this value is IID_IDirectInput8A or IID_IDirectInput8W. Passing the IID_IDirectInput8 define selects the ANSI or Unicode version of the interface, depending on whether UNICODE is defined during compilation.

ppvOut

Address of a pointer to a variable to receive the interface pointer if the call succeeds.

punkOuter

Pointer to the address of the controlling object's **IUnknown** interface for COM aggregation, or NULL if the interface is not aggregated. Most callers pass NULL. If aggregation is requested, the object returned in **ppvOut* is a pointer to **IUnknown**, as required by COM aggregation.

Return Values

If the function succeeds, the return value is DI_OK.

If the function fails, the return value can be one of the following error values:

DIERR_BETADIRECTINPUTVERSION
DIERR_INVALIDPARAM
DIERR_OLDDIRECTINPUTVERSION
DIERR_OUTOFMEMORY

Remarks

The DirectInput object created by this function is implemented in Dinput8d.dll. Versions of interfaces earlier than DirectX 8.0 cannot be obtained in this implementation. To use earlier versions, create the DirectInput object by using **DirectInputCreate** or **DirectInputCreateEx**, which are in Dinput.lib.

Calling the function with *punkOuter* = NULL is equivalent to creating the object through **CoCreateInstance**(*&CLSID_DirectInput8, punkOuter, CLSCTX_INPROC_SERVER, &IID_IDirectInput8W, lplpDirectInput)*, then initializing it with **IDirectInput8::Initialize**.

Calling the function with *punkOuter* != NULL is equivalent to creating the object through **CoCreateInstance**(*&CLSID_DirectInput8, punkOuter, CLSCTX_INPROC_SERVER, &IID_IUnknown, lplpDirectInput)*. The aggregated object must be initialized manually.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.
Import Library: Use Dinput8.lib.

Macros

This section describes the following macros used in Microsoft® DirectInput®.

- **DIDFT_GETINSTANCE**
- **DIDFT_GETTYPE**
- **DIDFT_MAKEINSTANCE**
- **DIEFT_GETTYPE**
- **DIMAKEUSAGEDWORD**
- **DISEQUENCE_COMPARE**

- **GET_DIDEVICE_SUBTYPE**
- **GET_DIDEVICE_TYPE**

Dinput.h also defines macros for C calls to all the methods of the **IDirectInput8** and **IDirectInputDevice8** interfaces. These macros eliminate the need for pointers to method tables. For example, the following is a C call to the **IDirectInputDevice8::Release** method.

```
lpdid->lpVtbl->Release(lpdid));
```

The equivalent macro call looks like this.

```
IDirectInputDevice8_Release(lpdid);
```

All these macros take the same parameters as the method calls themselves.

DIDFT_GETINSTANCE

The **DIDFT_GETINSTANCE** macro extracts the object instance number code from a data format type.

```
DIDFT_GETINSTANCE(n) LOWORD((n) >> 8)
```

Parameters

n

Microsoft® DirectInput® data format type. The possible values for this parameter are identical to those found in the **dwType** member of the **DIOBJECTDATAFORMAT** structure.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIDFT_MAKEINSTANCE, **DIDFT_GETTYPE**

DIDFT_GETTYPE

The **DIDFT_GETTYPE** macro extracts the object type code from a data format type.

```
DIDFT_GETTYPE(n)  LOBYTE(n)
```

Parameters

n

Microsoft® DirectInput® data format type. The possible values for this parameter are identical to those found in the **dwType** member of the **DIOBJECTDATAFORMAT** structure.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIDFT_GETINSTANCE

DIDFT_MAKEINSTANCE

The **DIDFT_MAKEINSTANCE** macro creates an instance identifier of a device object for packing in the **dwType** member of the **DIOBJECTDATAFORMAT** structure.

```
DIDFT_MAKEINSTANCE(n) ((WORD)(n) << 8)
```

Parameters

n

Instance of the object; for example, 1 for button 1 of a mouse.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIDFT_GETINSTANCE

DIEFT_GETTYPE

The **DIEFT_GETTYPE** macro extracts the effect type code from an effect format type.

```
DIEFT_GETTYPE(n)  LOBYTE(n)
```

Parameters

n

Microsoft® DirectInput® effect format type. The possible values for this parameter are identical to those found in the **dwEffType** member of the **DIEFFECTINFO** structure.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIMAKEUSAGEDWORD

The **DIMAKEUSAGEDWORD** macro combines the usage page and usage codes for a device object representing a Human Interface Device, for passing to the **IDirectInputDevice8::GetObjectInfo** method.

```
DWORD DIMAKEUSAGEDWORD(
  WORD UsagePage,
  WORD Usage
);
```

Parameters

UsagePage

Usage page of the device object.

Usage

Usage of the device object.

Remarks

This macro is declared in Dinput.h as follows:

```
#define DIMAKEUSAGEDWORD(UsagePage, Usage) \
    ((DWORD)MAKELONG(Usage, UsagePage))
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 98.
Header: Declared in Dinput.h.

DISEQUENCE_COMPARE

The **DISEQUENCE_COMPARE** macro compares two Microsoft® DirectInput® sequence numbers, compensating for wraparound.

```
DISEQUENCE_COMPARE(dwSequence1, cmp, dwSequence2) \
        ((int)((dwSequence1) - (dwSequence2)) cmp 0)
```

Parameters

dwSequence1
First sequence number to compare.

cmp
One of the following comparison operators: ==, !=, <, >, <=, or >=.

dwSequence2
Second sequence number to compare.

Return Values

Returns a nonzero value if the result of the comparison specified by the *cmp* parameter is true, or zero otherwise.

Remarks

The following example checks whether the *dwSequence1* parameter value precedes the *dwSequence2* parameter value chronologically.

```
BOOL Sooner = (DISEQUENCE_COMPARE(dwSequence1, <, dwSequence2));
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

GET_DIDEVICE_SUBTYPE

The **GET_DIDEVICE_SUBTYPE** macro extracts the device subtype code from a device type description code.

```
GET_DIDEVICE_SUBTYPE(dwDevType)  HIBYTE(dwDevType)
```

Parameters

dwDevType
Microsoft® DirectInput® device type description code. The possible values for this parameter are identical to those found in the **dwDevType** member of the **DIDEVICEINSTANCE** structure.

Remarks

The interpretation of the subtype code depends on the primary type.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

GET_DIDEVICE_TYPE, **DIDEVICEINSTANCE**

GET_DIDEVICE_TYPE

The **GET_DIDEVICE_TYPE** macro extracts the device primary type code from a device type description code.

```
GET_DIDEVICE_TYPE(dwDevType)  LOBYTE(dwDevType)
```

Parameters

dwDevType
Microsoft® DirectInput® device type description code. Possible values for this parameter are identical to those found in the **dwDevType** member of the **DIDEVICEINSTANCE** structure.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

GET_DIDEVICE_SUBTYPE, **DIDEVICEINSTANCE**

Structures

This section contains information on the following structures used with Microsoft® DirectInput®.

- **DIACTION**
- **DIACTIONFORMAT**
- **DICOLORSET**
- **DICONDITION**
- **DICONFIGUREDEVICESPARAMS**
- **DICONSTANTFORCE**
- **DICUSTOMFORCE**
- **DIDATAFORMAT**
- **DIDEVCAPS**
- **DIDEVICEIMAGEINFO**
- **DIDEVICEIMAGEINFOHEADER**
- **DIDEVICEINSTANCE**
- **DIDEVICEOBJECTDATA**
- **DIDEVICEOBJECTINSTANCE**
- **DIEFFECT**
- **DIEFFECTINFO**
- **DIEFFESCAPE**
- **DIENVELOPE**
- **DIFILEEFFECT**
- **DIJOYSTATE**
- **DIJOYSTATE2**
- **DIMOUSESTATE**
- **DIMOUSESTATE2**
- **DIOBJECTDATAFORMAT**
- **DIPERIODIC**
- **DIPROPCPOINTS**
- **DIPROPDWORD**
- **DIPROPGUIDANDPATH**
- **DIPROPHEADER**

- **DIPROPPOINTER**
- **DIPROPRANGE**
- **DIPROPSTRING**
- **DIRAMPFORCE**

Note

The memory for all DirectX structures must be initialized to 0 before use. In addition, all structures that contain a **dwSize** member should set the member to the size of the structure, in bytes, before use. The following example performs these tasks on a common structure, **DIDEVCAPS**:

```
DIDEVCAPS didevcaps; // Can't use this yet

ZeroMemory(&didevcaps, sizeof(didevcaps));
didevcaps.dwSize = sizeof(didevcaps);

// Now the structure Can be used.
.
.
```

CPOINT

Describes a calibration point. An array of **CPOINT** structures is contained by a **DIPROPCPOINTS** structure.

```
typedef struct _CPOINT
{
   LONG  lP;
   DWORD dwLog;
} CPOINT, *PCPOINT;
```

Members

lP
Raw data

dwLog
Scaling value to be associated with the **lP** number. The percentage of the maximum value multiplied by 10000.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

Using CPOINTs

DIACTION

Describes a mapping of one game action to one device semantic. An array of **DIACTION** structures is contained by a **DIACTIONFORMAT** structure.

```
typedef struct _DIACTION {
    UINT_PTR    uAppData;
    DWORD       dwSemantic;
    DWORD       dwFlags;
    union {
        LPCSTR  lptszActionName;
        UINT    uResIdString;
    };
    GUID        guidInstance;
    DWORD       dwObjID;
    DWORD       dwHow;
} DIACTION, *LPDIACTION;

typedef const DIACTION *LPCDIACTION;
```

Members

uAppData

Address of an application-defined **UINT** value to be returned to the application by **IDirectInputDevice8::GetDeviceData** when the state of the control associated with the action changes. This value is returned in the **uAppData** member of the **DIDEVICEOBJECTDATA** structure. It is typically an identifier for the application-specific action associated with the device object, but can also be a function pointer.

dwSemantic

For a joystick, a predefined action mapping constant for this application genre representing a virtual control or a constant for a control not defined in the genre. For a keyboard, mouse, or Microsoft® DirectPlay® voice device, a constant that represents a particular device object. See Action Mapping Constants.

dwFlags

Flags used to request specific attributes or processing. Can be zero or one or more of the following values.

DIA_APPFIXED

The action cannot be remapped by Microsoft DirectInput®.

DIA_APPMAPPED

The **dwObjID** member is valid, and **IDirectInputDevice8::BuildActionMap** will not override the application-defined mapping.

DIA_APPNOMAP
This action is not to be mapped.

DIA_FORCEFEEDBACK
The action must be mapped to an actuator or trigger.

DIA_NORANGE
The default range is not to be set for this action. This flag can be set only for absolute axis actions.

lptszActionName
Application-defined name of the action. This string is displayed by the device property sheet when **IDirectInput8::ConfigureDevices** is called.

uResIdString
Resource identifier for the string for this action. The module instance for this resource is specified in the **hInstString** member of the **DIACTIONFORMAT** structure that contains this structure.

guidInstance
Device instance GUID if a specific device is requested. Otherwise GUID_NULL.

dwObjID
Control identifier. Use the **DIDFT_GETINSTANCE** and **DIDFT_GETTYPE** macros to retrieve the instance and type from this value.

dwHow
When the structure is returned by **IDirectInputDevice8::BuildActionMap**, this member receives a value to indicate the mapping mechanism used by DirectInput to configure the action. The following values are defined.

DIAH_APPREQUESTED
The mapping was configured by the application, which specified the device (**guidInstance**) and device object (**dwObjID**) when calling **BuildActionMap**.

DIAH_DEFAULT
The mapping was determined by DirectInput in the absence of other mapping information.

DIAH_ERROR
An error occurred. The action cannot be matched to a control on the device. The action will be ignored when the action map is set.

DIAH_HWAPP
The mapping was specified by the hardware manufacturer for this game.

DIAH_HWDEFAULT
The mapping was specified by the hardware manufacturer for this genre.

DIAH_UNMAPPED
No suitable device object was found.

DIAH_USERCONFIG
The mapping was configured by the user.

This member is ignored when the action map is passed to **IDirectInputDevice8::SetActionMap**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInput8::EnumDevicesBySemantics, **IDirectInputDevice8::SetActionMap**, **IDirectInput8::ConfigureDevices**, **IDirectInputDevice8::BuildActionMap**, Action Mapping

DIACTIONFORMAT

Contains information about the calling application and acts as a container for an array of **DIACTION** structures that define action-to-control mappings for a genre.

```
typedef struct {
    DWORD           dwSize;
    DWORD           dwActionSize;
    DWORD           dwDataSize;
    DWORD           dwNumActions;
    LPDIACTION      rgoAction;
    GUID            guidActionMap;
    DWORD           dwGenre;
    DWORD           dwBufferSize;
    OPTIONAL LONG   lAxisMin;
    OPTIONAL LONG   lAxisMax;
    OPTIONAL HINSTANCE hInstString;
    FILETIME        ftTimeStamp;
    DWORD           dwCRC;
    TCHAR           tszActionMap[MAX_PATH];
} DIACTIONFORMAT;
```

Members

dwSize
DWORD value that specifies the size of the **DIACTIONFORMAT** structure, in bytes.

dwActionSize
DWORD value that specifies the size of the **DIACTION** structure, in bytes.

dwDataSize
DWORD value that specifies the size of the device data to be returned by the device for immediate device data, in bytes. This member should be **dwNumActions** multiplied by four.

dwNumActions

DWORD value that specifies the number of elements in the **rgoAction** array.

rgoAction

Address of an array of **DIACTION** structures, each of which describes how an action maps to a virtual control or device object, and how the mapping information should be displayed to the user.

guidActionMap

GUID that identifies the action map. Device manufacturers can use this value to tune mappings for a specific title.

dwGenre

DWORD value that specifies the genre of the application. For possible values, see Action Mapping Constants.

dwBufferSize

DWORD value that specifies the number of input data packets in the buffer for each device to which this action map is to be applied. The buffer size must be set to a value greater than zero in order to retrieve data by using **IDirectInputDevice8::GetDeviceData**. Applications that set this member to the desired buffer size before calling **IDirectInputDevice8::SetActionMap** do not need to set the DIPROP_BUFFERSIZE property by using **IDirectInputDevice8::SetProperty**.

lAxisMin

Minimum value for the range of scaled data to be returned for all axes. This value is ignored for a specific action axis if the DIA_NORANGE flag is set in **DIACTION.dwFlags**.

This value is valid only for axis actions and should be set to zero for all other actions. It is used as the **DIPROPRANGE.lMin** value to set the range property on an absolute axis when the action map is applied using **IDirectInputDevice8::SetActionMap**.

lAxisMax

Maximum value for the range of scaled data to be returned for all axes. This value is ignored for a specific action axis if the DIA_NORANGE flag is set in **DIACTION.dwFlags**.

This value is valid only for axis actions and should be set to zero for all other actions. It is used as the **DIPROPRANGE.lMax** value to set the range property on an absolute axis when the action map is applied using **IDirectInputDevice8::SetActionMap**.

hInstString

Handle of the module containing string resources for action names, as specified in the **uResIdString** member of the **DIACTION** structure for each action. Can be zero if action names are specified in the **lptszActionName** member of the **DIACTION** structure for each action.

ftTimeStamp

FILETIME structure that receives the time at which this action map was last written to disk. See Remarks.

dwCRC

Cyclic redundancy check for this map. Used internally by Microsoft® DirectInput® to determine when a set of mappings should be saved to disk.

tszActionMap

Null-terminated string, of maximum length MAX_PATH, that specifies the friendly name for this action map. This string appears in the drop-down list box in the default property sheet.

Remarks

The **ftTimeStamp** member can contain special values that apply to new and unused devices. New devices have never been enumerated for this application and have never had an action map applied to them. Unused devices have been enumerated for the application before but have never had an action map applied. New devices always have DIAFTS_NEWDEVICELOW and DIAFTS_NEWDEVICEHIGH in the low and high **DWORD**s respectively of the **FILETIME** structure. Unused devices have DIAFTS_UNUSEDDEVICELOW and DIAFTS_UNUSEDDEVICEHIGH in these positions.

Applications should not use **ftTimeStamp** to check for new devices. Instead, they should look for the DIEDBS_RECENTDEVICE and DIEDBS_NEWDEVICE flags in the enumeration callback. For more information, see **DIEnumDevicesBySemanticsCallback**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInput8::EnumDevicesBySemantics, **IDirectInputDevice8::SetActionMap**, **IDirectInput8::ConfigureDevices**, **IDirectInputDevice8::BuildActionMap**, Action Mapping

DICOLORSET

Describes a set of colors used to draw the device configuration user interface. It is part of the **DICONFIGUREDEVICESPARAMS** structure.

```
typedef struct _DICOLORSET {
    DWORD    dwSize
    D3DCOLOR cTextFore;
    D3DCOLOR cTextHighlight;
    D3DCOLOR cCalloutLine;
    D3DCOLOR cCalloutHighlight;
```

```
    D3DCOLOR cBorder;
    D3DCOLOR cControlFill;
    D3DCOLOR cHighlightFill;
    D3DCOLOR cAreaFill;
} DICOLORSET, *LPDICOLORSET;

typedef const DICOLORSET *LPCDICOLORSET;
```

Members

dwSize
Size of this structure, in bytes.

cTextFore
Foreground text color.

cTextHighlight
Foreground color for highlighted text.

cCalloutLine
Color used to display callout lines.

cCalloutHighlight
Color used to display callout lines.

cBorder
Border color, used to display lines around controls such as tabs and buttons.

cControlFill
Fill color for controls such as tabs and buttons.

cHighlightFill
Fill color for highlighted controls.

cAreaFill
Fill color for areas outside controls such as tabs and buttons.

Remarks

Text background color is always transparent.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

D3DCOLOR

DICONDITION

Contains type-specific information for effects that are marked as DIEFT_CONDITION.

A pointer to an array of **DICONDITION** structures for an effect is passed in the **lpvTypeSpecificParams** member of the **DIEFFECT** structure. The number of elements in the array must be either one, or equal to the number of axes associated with the effect.

```
typedef struct DICONDITION {
    LONG  lOffset;
    LONG  lPositiveCoefficient;
    LONG  lNegativeCoefficient;
    DWORD dwPositiveSaturation;
    DWORD dwNegativeSaturation;
    LONG  lDeadBand;
} DICONDITION, *LPDICONDITION;

typedef const DICONDITION *LPCDICONDITION;
```

Members

lOffset
Offset for the condition, in the range from –10,000 through 10,000.

lPositiveCoefficient
Coefficient constant on the positive side of the offset, in the range from –10,000 through 10,000.

lNegativeCoefficient
Coefficient constant on the negative side of the offset, in the range from –10,000 through 10,000.

If the device does not support separate positive and negative coefficients, the value of **lNegativeCoefficient** is ignored, and the value of **lPositiveCoefficient** is used as both the positive and negative coefficients.

dwPositiveSaturation
Maximum force output on the positive side of the offset, in the range from 0 through 10,000.

If the device does not support force saturations, the value of this member is ignored.

dwNegativeSaturation
Maximum force output on the negative side of the offset, in the range from 0 through 10,000.

If the device does not support force saturation, the value of this member is ignored.

If the device does not support separate positive and negative saturation, the value of **dwNegativeSaturation** is ignored, and the value of **dwPositiveSaturation** is used as both the positive and negative saturation.

lDeadBand

Region around **lOffset** in which the condition is not active, in the range from 0 through 10,000. In other words, the condition is not active between **lOffset** minus **lDeadBand** and **lOffset** plus **lDeadBand**.

Remarks

Different types of conditions interpret the parameters differently, but the basic idea is that force resulting from a condition is equal to *A(q - q0)* where *A* is a scaling coefficient, *q* is some metric, and *q0* is the neutral value for that metric.

The preceding simplified formula must be adjusted if a nonzero deadband is provided. If the metric is less than **lOffset** - **lDeadBand**, the resulting force is given by the following formula:

force = **lNegativeCoefficient** * (*q* - (**lOffset** - **lDeadBand**))

Similarly, if the metric is greater than **lOffset** + **lDeadBand**, the resulting force is given by the following formula:

force = **lPositiveCoefficient** * (*q* - (**lOffset** + **lDeadBand**))

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.

If the number of **DICONDITION** structures in the array is equal to the number of axes for the effect, the first structure applies to the first axis, the second applies to the second axis, and so on. For example, a two-axis spring condition with **lOffset** set to 0 in both **DICONDITION** structures would have the same effect as the joystick self-centering spring. When a condition is defined for each axis in this way, the effect must not be rotated.

If there is a single **DICONDITION** structure for an effect with more than one axis, the direction along which the parameters of the **DICONDITION** structure are in effect is determined by the direction parameters passed in the **rglDirection** field of the **DIEFFECT** structure. For example, a friction condition rotated 45 degrees (in polar coordinates) would resist joystick motion in the northeast-southwest direction but would have no effect on joystick motion in the northwest-southeast direction.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DICONFIGUREDEVICESPARAMS

Contains information for the device configuration property sheet. It is used by the **IDirectInput8::ConfigureDevices** method.

```
typedef struct _DICONFIGUREDEVICESPARAMS {
    DWORD           dwSize;
    DWORD           dwcUsers;
    LPSTR           lptszUserNames;
    DWORD           dwcFormats;
    LPDIACTIONFORMAT lprgFormats;
    HWND            hwnd;
    DICOLORSET      dics;
    IUnknown FAR *   lpUnkDDSTarget;
} DICONFIGUREDEVICESPARAMS, *LPDICONFIGUREDEVICESPARAMS;
```

Members

dwSize
Size of this structure, in bytes.

dwcUsers
Count of user names in the buffer at lptszUserNames. If lptszUserNames is NULL (to indicate that default user names should be used), the value in this member is ignored. If the dwcUsers value exceeds the number of entries actually in the buffer, the method fails, returning DIERR_INVALIDPARAM.

lptszUserNames
Pointer to a buffer containing a series of null-terminated name strings, the final element being designated by a double-null terminator. The lptszUserNames parameter can be set to NULL to instruct the use of default names. If the application passes more names than the dwcUsers count indicates, only the names within the count are used. If an application specifies names that are different from the names currently assigned to devices, ownership is revoked for all devices, a default name is created for the mismatched name, and the interface shows **(No User)** for all devices.

dwcFormats
Count of structures in the array at **lprgFormats**.

lprgFormats
Pointer to an array of **DIACTIONFORMAT** structures that contains action-mapping information for each genre used by the game. On input, this array contains action-to-control mappings and strings to display as callouts for each mapping. The configuration interface displays the genres in its drop-down list in the same order as in the array.

hwnd
Handle to the top-level window of the calling application. The member is needed only for applications that run in windowed mode, and is otherwise ignored.

dics

DICOLORSET structure that describes the color scheme to apply to the configuration user interface. Passing a zero-initialized DICOLORSET structure causes the default color scheme to be used.

lpUnkDDSTarget

Pointer to the **IUnknown** interface for a Microsoft® DirectDraw® or Microsoft Direct3D® target surface object that will be filled to contain an image of the configuration user interface. The image represents the current state of the user interface at the time that the **DIEnumDevicesBySemanticsCallback** function is invoked. The target surface will retain any alpha information passed as part of the DICOLORSET structure above, and also any alpha information encoded into the device image by the hardware manufacturer. The target surface object referred to by the IUnknown interface must support either **IDirect3DSurface8** or **IDirectDrawSurface7**. Full-screen applications using DirectDraw must have target surfaces created using the DDSCAPS_SYSTEMEMORY flag.

Applications that are not using DirectDraw and applications that are using DirectDraw but are windowed rather than using the full screen may pass NULL in this parameter. This will cause Microsoft DirectInput® to use the Windows Graphics Device Interface (GDI) functions to draw the configuration user interface image.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DICONSTANTFORCE

Contains type-specific information for effects that are marked as DIEFT_CONSTANTFORCE.

The structure describes a constant force effect.

A pointer to a single **DICONSTANTFORCE** structure for an effect is passed in the **lpvTypeSpecificParams** member of the **DIEFFECT** structure.

```
typedef struct DICONSTANTFORCE {
    LONG lMagnitude;
} DICONSTANTFORCE, *LPDICONSTANTFORCE;

typedef const DICONSTANTFORCE *LPCDICONSTANTFORCE;
```

Members

lMagnitude

The magnitude of the effect, in the range from –10,000 through 10,000. If an envelope is applied to this effect, the value represents the magnitude of the sustain. If no envelope is applied, the value represents the amplitude of the entire effect.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DICUSTOMFORCE

Contains type-specific information for effects that are marked as DIEFT_CUSTOMFORCE.

The structure describes a custom or user-defined force.

A pointer to a **DICUSTOMFORCE** structure for an effect is passed in the **lpvTypeSpecificParams** member of the **DIEFFECT** structure.

```
typedef struct DICUSTOMFORCE {
    DWORD  cChannels;
    DWORD  dwSamplePeriod;
    DWORD  cSamples;
    LPLONG rglForceData;
} DICUSTOMFORCE, *LPDICUSTOMFORCE;

typedef const DICUSTOMFORCE *LPCDICUSTOMFORCE;
```

Members

cChannels
Number of channels (axes) affected by this force.

The first channel is applied to the first axis associated with the effect, the second to the second, and so on. If there are fewer channels than axes, nothing is associated with the extra axes.

If there is only a single channel, the effect is rotated in the direction specified by the **rglDirection** member of the **DIEFFECT** structure. If there is more than one channel, rotation is not allowed.

Not all devices support rotation of custom effects.

dwSamplePeriod
Sample period, in microseconds.

cSamples
Total number of samples in the **rglForceData**. It must be an integral multiple of the **cChannels**.

rglForceData
Pointer to an array of force values representing the custom force. If multiple channels are provided, the values are interleaved. For example, if **cChannels** is 3, the first element of the array belongs to the first channel, the second to the second, and the third to the third.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIDATAFORMAT

Describes a device's data format. This structure is used with the **IDirectInputDevice8::SetDataFormat** method.

```
typedef struct DIDATAFORMAT {
    DWORD dwSize;
    DWORD dwObjSize;
    DWORD dwFlags;
    DWORD dwDataSize;
    DWORD dwNumObjs;
    LPDIOBJECTDATAFORMAT rgodf;
} DIDATAFORMAT, *LPDIDATAFORMAT;

typedef const DIDATAFORMAT *LPCDIDATAFORMAT;
```

Members

dwSize
Size of this structure, in bytes.

dwObjSize
Size of the **DIOBJECTDATAFORMAT** structure, in bytes.

dwFlags
Flags describing other attributes of the data format. This value can be one of the following:

DIDF_ABSAXIS
The axes are in absolute mode. Setting this flag in the data format is equivalent to manually setting the axis mode property, using the **IDirectInputDevice8::SetProperty** method. This cannot be combined with DIDF_RELAXIS flag.

DIDF_RELAXIS
The axes are in relative mode. Setting this flag in the data format is equivalent to manually setting the axis mode property using the

IDirectInputDevice8::SetProperty method. This cannot be combined with the DIDF_ABSAXIS flag.

dwDataSize
Size of a data packet returned by the device, in bytes. This value must be a multiple of 4 and must exceed the largest offset value for an object's data within the data packet.

dwNumObjs
Number of objects in the **rgodf** array.

rgodf
Address to an array of **DIOBJECTDATAFORMAT** structures. Each structure describes how one object's data should be reported in the device data. Typical errors include placing two pieces of information in the same location and placing one piece of information in more than one location.

Remarks

Applications do not typically need to create a **DIDATAFORMAT** structure. An application can use one of the predefined global data format variables, *c_dfDIMouse*, *c_dfDIMouse2*, c*_dfDIKeyboard*, *c_dfDIJoystick*, or *c_dfDIJoystick2*.

The following code example sets a data format that can be used by applications that need two axes (reported in absolute coordinates) and two buttons:

```
// Suppose an application uses the following
// structure to read device data.

typedef struct MYDATA {
    LONG  lX;               // X-axis goes here.
    LONG  lY;               // Y-axis goes here.
    BYTE  bButtonA;             // One button goes here.
    BYTE  bButtonB;             // Another button goes here.
    BYTE  bPadding[2];          // Must be DWORD multiple in size.
} MYDATA;

// Then, it can use the following data format.

DIOBJECTDATAFORMAT rgodf[ ] = {
  { &GUID_XAxis, FIELD_OFFSET(MYDATA, lX),
    DIDFT_AXIS | DIDFT_ANYINSTANCE, 0, },
  { &GUID_YAxis, FIELD_OFFSET(MYDATA, lY),
    DIDFT_AXIS | DIDFT_ANYINSTANCE, 0, },
  { &GUID_Button, FIELD_OFFSET(MYDATA, bButtonA),
    DIDFT_BUTTON | DIDFT_ANYINSTANCE, 0, },
  { &GUID_Button, FIELD_OFFSET(MYDATA, bButtonB),
    DIDFT_BUTTON | DIDFT_ANYINSTANCE, 0, },
};
```

```
#define numObjects (sizeof(rgodf) / sizeof(rgodf[0]))

DIDATAFORMAT df = {
    sizeof(DIDATAFORMAT),       // Size of this structure
    sizeof(DIOBJECTDATAFORMAT), // Size of object data format
    DIDF_ABSAXIS,               // Absolute axis coordinates
    sizeof(MYDATA),             // Size of device data
    numObjects,                 // Number of objects
    rgodf,                      // And here they are
};
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIACTIONFORMAT

DIDEVCAPS

Describes a Microsoft® DirectInput® device's capabilities. This structure is used with the **IDirectInputDevice8::GetCapabilities** method.

```
typedef struct DIDEVCAPS {
    DWORD dwSize;
    DWORD dwFlags;
    DWORD dwDevType;
    DWORD dwAxes;
    DWORD dwButtons;
    DWORD dwPOVs;
    DWORD dwFFSamplePeriod;
    DWORD dwFFMinTimeResolution;
    DWORD dwFirmwareRevision;
    DWORD dwHardwareRevision;
    DWORD dwFFDriverVersion;
} DIDEVCAPS, *LPDIDEVCAPS;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized by the application before a call to the **IDirectInputDevice8::GetCapabilities** method.

dwFlags

Flags associated with the device. This value can be a combination of the following:

DIDC_ALIAS
The device is a duplicate of another DirectInput device. Alias devices are by default not enumerated by **IDirectInput8::EnumDevices**.

DIDC_ATTACHED
The device is physically attached.

DIDC_DEADBAND
The device supports deadband for at least one force-feedback condition.

DIDC_EMULATED
If this flag is set, the data is coming from a user mode device interface, such as a Human Interface Device (HID), or by some other ring 3 means. If it is not set, the data is coming directly from a kernel mode driver.

DIDC_FORCEFEEDBACK
The device supports force feedback.

DIDC_FFFADE
The force-feedback system supports the fade parameter for at least one effect. If the device does not support fade, the fade level and fade time members of the **DIENVELOPE** structure are ignored by the device.

After a call to the **IDirectInputDevice8::GetEffectInfo** method, an individual effect sets the DIEFT_FFFADE flag if fade is supported for that effect.

DIDC_FFATTACK
The force-feedback system supports the attack parameter for at least one effect. If the device does not support attack, the attack level and attack time members of the **DIENVELOPE** structure are ignored by the device.

After a call to the **IDirectInputDevice8::GetEffectInfo** method, an individual effect sets the DIEFT_FFATTACK flag if attack is supported for that effect.

DIDC_HIDDEN
Fictitious device created by a device driver so that it can generate keyboard and mouse events. Such devices are not normally enumerated by **IDirectInput8::EnumDevices** or configured by **IDirectInput8::ConfigureDevices**.

DIDC_PHANTOM
Placeholder for a device that might exist in the future. Phantom devices are by default not enumerated by **IDirectInput8::EnumDevices**.

DIDC_POLLEDDATAFORMAT
At least one object in the current data format is polled, rather than interrupt-driven. For these objects, the application must explicitly call the **IDirectInputDevice8::Poll** method to obtain data.

DIDC_POLLEDDEVICE
At least one object on the device is polled, rather than interrupt-driven. For these objects, the application must explicitly call the

IDirectInputDevice8::Poll method to obtain data. HID devices can contain a mixture of polled and nonpolled objects.

DIDC_POSNEGCOEFFICIENTS
The force-feedback system supports two coefficient values for conditions (one for the positive displacement of the axis and one for the negative displacement of the axis) for at least one condition. If the device does not support both coefficients, the negative coefficient in the **DICONDITION** structure is ignored.

After a call to the **IDirectInputDevice8::GetEffectInfo** method, an individual condition sets the DIEFT_POSNEGCOEFFICIENTS flag if separate positive and negative coefficients are supported for that condition.

DIDC_POSNEGSATURATION
The force-feedback system supports a maximum saturation for both positive and negative force output for at least one condition. If the device does not support both saturation values, the negative saturation in the **DICONDITION** structure is ignored.

After a call to the **IDirectInputDevice8::GetEffectInfo** method, an individual condition sets the DIEFT_POSNEGSATURATION flag if separate positive and negative saturation are supported for that condition.

DIDC_SATURATION
The force-feedback system supports the saturation of condition effects for at least one condition. If the device does not support saturation, the force generated by a condition is limited only by the maximum force that the device can generate.

After a call to the **IDirectInputDevice8::GetEffectInfo** method, an individual condition sets the DIEFT_SATURATION flag if saturation is supported for that condition.

DIDC_STARTDELAY
The force-feedback system supports the start delay parameter for at least one effect. If the device does not support start delays, the **dwStartDelay** member of the **DIEFFECT** structure is ignored.

dwDevType
Device type specifier. This member can contain values identical to those in the **dwDevType** member of the **DIDEVICEINSTANCE** structure.

dwAxes
Number of axes available on the device.

dwButtons
Number of buttons available on the device.

dwPOVs
Number of point-of-view controllers available on the device.

dwFFSamplePeriod
Minimum time between playback of consecutive raw force commands, in microseconds.

dwFFMinTimeResolution

Minimum time, in microseconds, that the device can resolve. The device rounds any times to the nearest supported increment. For example, if the value of **dwFFMinTimeResolution** is 1000, the device would round any times to the nearest millisecond.

dwFirmwareRevision
Firmware revision of the device.

dwHardwareRevision
Hardware revision of the device.

dwFFDriverVersion
Version number of the device driver.

Remarks

The semantics of version numbers are left to the manufacturer of the device. The only guarantee is that newer versions have larger numbers.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIDEVICEINSTANCE

DIDEVICEIMAGEINFO

Carries information required to display a device image or an overlay image with a callout. This structure is passed to the **IDirectInputDevice8::GetImageInfo** method as an array within a **DIDEVICEIMAGEINFOHEADER** structure.

```
typedef struct _DIDDEVICEIMAGEINFO {
    TCHAR  tszImagePath[MAX_PATH];
    DWORD  dwFlags;
    // The following are valid if DIDIFT_OVERLAY is present
    // in dwFlags
    DWORD  dwViewID;
    RECT   rcOverlay;
    DWORD  dwObjID;
    DWORD  dwcValidPts;
    POINT  rgptCalloutLine[5];
    RECT   rcCalloutRect;
    DWORD  dwTextAlign;
} DIDEVICEIMAGEINFO, * LPDIDEVICEIMAGEINFO;
```

Members

tszImagePath
Fully qualified path to the file that contains an image of the device. The file format is given in **dwFlags**. If no image is available for the device, this member will be set to NULL. If so, the application is responsible for enumerating controls on the device and displaying a default listing of actions to device controls (similar to the method used by most applications before Microsoft® DirectX® 8.0).

dwFlags
Flag that describes the intended use of the image.
DIDIFT_CONFIGURATION
The file is used to display the current configuration of actions on the device. Overlay image coordinates are relative to the upper-left corner of the configuration image.
DIDIFT_OVERLAY
The file (if provided) is an overlay for a configuration image. The **dwViewID**, **rcOverlay**, **dwObjID**, **rgptCalloutLine**, **rcCalloutRect**, and **dwTextAlign** members are valid and contain data used to display the overlay and callout information for a single control on the device. If no file is provided (null path string), all other pertinent members are relevant except **rcOverlay**.

dwViewID
For device view images (DIDIFT_CONFIGURATION), this is the ID of the device view. For device control overlays (DIDIFT_OVERLAY), this value refers to the device view (by ID) over which an image and callout information should be displayed.

rcOverlay
Rectangle, using coordinates relative to the top-left pixel of the device configuration image, in which the overlay image should be painted. This member is valid only if the DIDIFT_OVERLAY flag is present in **dwFlags**.

dwObjID
Control identifier, as a combination of DIDFT_* flags and an instance value, to which an overlay image corresponds for this device. Applications use the DIDFT_GETINSTANCE and DIDFT_GETTYPE macros to decode this value to its constituent parts. This member is valid only if the DIDIFT_OVERLAY flag is present in **dwFlags**.

dwcValidPts
Number of points in the array at **rgptCalloutLine**.

rgptCalloutLine
Array of **POINT** structures that specify coordinates of the points describing a callout line. A callout line connects a device control to a caption for the game action. Each line can have from one to four segments. This member is valid only if the DIDIFT_OVERLAY flag is present in **dwFlags**.

rcCalloutRect
RECT structure that describes the rectangle in which the game action string is displayed. If the string cannot fit within the rectangle, the application is

responsible for handling clipping. This member is valid only if the DIDIFT_OVERLAY flag is present in **dwFlags**.

dwTextAlign

DWORD value that specifies the alignment of the text in the rectangle described by the **rcCalloutRect** member. Must be one horizontal alignment flag combined with one vertical alignment flag.

The following horizontal alignment flags are defined.

DIDAL_LEFTALIGNED
Text is aligned on the left border.

DIDAL_CENTERED
Text is horizontally centered.

DIDAL_RIGHTALIGNED
Text is aligned on the right border.

The following vertical alignment flags are defined.

DIDAL_MIDDLE
The text is vertically centered.

DIDAL_TOPALIGNED
The text is aligned on the top border.

DIDAL_BOTTOMALIGNED
The text is aligned on the bottom border.

This member is valid only if the DIDIFT_OVERLAY flag is present in **dwFlags**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIDEVICEIMAGEINFOHEADER

Contains information about device images. Used in the **IDirectInputDevice8::GetImageInfo** method.

```
typedef struct _DIDEVICEIMAGEINFOHEADER{
    DWORD  dwSize;
    DWORD  dwSizeImageInfo;
    DWORD  dwcViews;
    DWORD  dwcButtons;
    DWORD  dwcAxes;
    DWORD  dwcPOVs;
    DWORD  dwBufferSize;
    DWORD  dwBufferUsed;
    DIDEVICEIMAGEINFO *lprgImageInfoArray;
```

```
} DIDEVICEIMAGEINFOHEADER, * LPDIDEVICEIMAGEINFOHEADER;
```

Members

dwSize
DWORD value that specifies the size of this structure, in bytes. Must be initialized before the structure can be used.

dwSizeImageInfo
DWORD value that specifies the size of the **DIDEVICEIMAGEINFO** structure, in bytes. Must be initialized before this structure can be used.

dwcViews
DWORD variable that receives the number of views of this device.

dwcButtons
DWORD variable that receives the number of buttons on the device.

dwcAxes
DWORD variable that receives the number of axes on the device.

dwcPOVs
DWORD variable that receives the number of point-of-view controllers on the device.

dwBufferSize
DWORD value that specifies the size, in bytes, of the buffer at **lprgImageInfo**. When set to zero, the **IDirectInputDevice8::GetImageInfo** method ignores all other members and returns the minimum buffer size required to hold information for all images.

dwBufferUsed
DWORD value that receives the size, in bytes, of the memory used in the buffer at **lprgImageInfo**. When **dwBufferSize** is set to zero, the **IDirectInputDevice8::GetImageInfo** method sets this member to the minimum size needed to hold information for all images.

lprgImageInfoArray
Pointer to a buffer that receives an array of **DIDEVICEIMAGEINFO** structures describing all the device images and views, overlay images, and callout-string coordinates.

Remarks

The buffer at **lprgImageInfoArray** must be large enough to hold all required image information structures. Applications can query for the required size by calling the **IDirectInputDevice8::GetImageInfo** method with the **dwBufferSize** member set to zero. After the call, **dwBufferUsed** contains the amount of memory, in bytes, that was modified.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIDEVICEINSTANCE

Describes an instance of a Microsoft® DirectInput® device. This structure is used with the **IDirectInput8::EnumDevices**, **IDirectInput8::EnumDevicesBySemantics**, and **IDirectInputDevice8::GetDeviceInfo** methods.

```
typedef struct DIDEVICEINSTANCE {
    DWORD dwSize;
    GUID  guidInstance;
    GUID  guidProduct;
    DWORD dwDevType;
    TCHAR tszInstanceName[MAX_PATH];
    TCHAR tszProductName[MAX_PATH];
    GUID  guidFFDriver;
    WORD  wUsagePage;
    WORD  wUsage;
} DIDEVICEINSTANCE, *LPDIDEVICEINSTANCE;

typedef const DIDEVICEINSTANCE  *LPCDIDEVICEINSTANCE;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before the structure is used.

guidInstance
Unique identifier for the instance of the device. An application can save the instance GUID into a configuration file and use it at a later time. Instance GUIDs are specific to a particular computer. An instance GUID obtained from one computer is unrelated to instance GUIDs on another.

guidProduct
Unique identifier for the product. This identifier is established by the manufacturer of the device.

dwDevType
Device type specifier. The least-significant byte of the device type description code specifies the device type. The next-significant byte specifies the device subtype. This value can also be combined with DIDEVTYPE_HID, which specifies a Human Interface Device (HID).

tszInstanceName[MAX_PATH]

Friendly name for the instance. For example, "Joystick 1."

tszProductName[MAX_PATH]
Friendly name for the product.

guidFFDriver
Unique identifier for the driver being used for force feedback. The driver's manufacturer establishes this identifier.

wUsagePage
If the device is a Human Interface Device (HID), this member contains the HID usage page code.

wUsage
If the device is a Human Interface Device (HID), this member contains the HID usage code.

Remarks

The following device types and subtypes are defined for use in the **dwDevType** member.

DI8DEVTYPE_1STPERSON
First-person action game device. The following subtypes are defined.

DI8DEVTYPE1STPERSON_LIMITED
Device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPE1STPERSON_SHOOTER
Device designed for first-person shooter games.

DI8DEVTYPE1STPERSON_SIXDOF
Device with six degrees of freedom; that is, three lateral axes and three rotational axes.

DI8DEVTYPE1STPERSON_UNKNOWN
Unknown subtype.

DI8DEVTYPE_DEVICE
Device that does not fall into another category.

DI8DEVTYPE_DEVICECTRL
Input device used to control another type of device from within the context of the application. The following subtypes are defined.

DI8DEVTYPEDEVICECTRL_COMMSSELECTION
Control used to make communications selections.

DI8DEVTYPEDEVICECTRL_COMMSSELECTION_HARDWIRED
Device that must use its default configuration and cannot be remapped.

DI8DEVTYPEDEVICECTRL_UNKNOWN
Unknown subtype.

DI8DEVTYPE_DRIVING
Device for steering. The following subtypes are defined.

DI8DEVTYPEDRIVING_COMBINEDPEDALS
Steering device that reports acceleration and brake pedal values from a single axis.

DI8DEVTYPEDRIVING_DUALPEDALS
Steering device that reports acceleration and brake pedal values from separate axes.

DI8DEVTYPEDRIVING_HANDHELD
Hand-held steering device.

DI8DEVTYPEDRIVING_LIMITED
Steering device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEDRIVING_THREEPEDALS
Steering device that reports acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPE_FLIGHT
Controller for flight simulation. The following subtypes are defined.

DI8DEVTYPEFLIGHT_LIMITED
Flight controller that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEFLIGHT_RC
Flight device based on a remote control for model aircraft.

DI8DEVTYPEFLIGHT_STICK
Joystick.

DI8DEVTYPEFLIGHT_YOKE
Yoke.

DI8DEVTYPE_GAMEPAD
Gamepad. The following subtypes are defined.

DI8DEVTYPEGAMEPAD_LIMITED
Gamepad that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_STANDARD
Standard gamepad that provides the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_TILT
Gamepad that can report x-axis and y-axis data based on the attitude of the controller.

DI8DEVTYPE_JOYSTICK
Joystick. The following subtypes are defined.

DI8DEVTYPEJOYSTICK_LIMITED
Joystick that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEJOYSTICK_STANDARD
Standard joystick that provides the minimum number of device objects for action mapping.

DI8DEVTYPE_KEYBOARD
Keyboard or keyboard-like device. The following subtypes are defined.

DI8DEVTYPEKEYBOARD_UNKNOWN
Subtype could not be determined.

DI8DEVTYPEKEYBOARD_PCXT
IBM PC/XT 83-key keyboard.

DI8DEVTYPEKEYBOARD_OLIVETTI
Olivetti 102-key keyboard.

DI8DEVTYPEKEYBOARD_PCAT
IBM PC/AT 84-key keyboard.

DI8DEVTYPEKEYBOARD_PCENH
IBM PC Enhanced 101/102-key or Microsoft Natural® keyboard.

DI8DEVTYPEKEYBOARD_NOKIA1050
Nokia 1050 keyboard.

DI8DEVTYPEKEYBOARD_NOKIA9140
Nokia 9140 keyboard.

DI8DEVTYPEKEYBOARD_NEC98
Japanese NEC PC98 keyboard.

DI8DEVTYPEKEYBOARD_NEC98LAPTOP
Japanese NEC PC98 laptop keyboard.

DI8DEVTYPEKEYBOARD_NEC98106
Japanese NEC PC98 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPAN106
Japanese 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPANAX
Japanese AX keyboard.

DI8DEVTYPEKEYBOARD_J3100
Japanese J3100 keyboard.

DI8DEVTYPE_MOUSE
A mouse or mouse-like device (such as a trackball). The following subtypes are defined.

DI8DEVTYPEMOUSE_ABSOLUTE
Mouse that returns absolute axis data.

DI8DEVTYPEMOUSE_FINGERSTICK
Fingerstick.

DI8DEVTYPEMOUSE_TOUCHPAD
Touchpad.

DI8DEVTYPEMOUSE_TRACKBALL

Trackball.

DI8DEVTYPEMOUSE_TRADITIONAL
Traditional mouse.

DI8DEVTYPEMOUSE_UNKNOWN
Subtype could not be determined.

DI8DEVTYPE_REMOTE
Remote-control device. The following subtype is defined.

DI8DEVTYPEREMOTE_UNKNOWN
Subtype could not be determined.

DI8DEVTYPE_SCREENPOINTER
Screen pointer. The following subtypes are defined.

DI8DEVTYPESCREENPTR_UNKNOWN
Unknown subtype.

DI8DEVTYPESCREENPTR_LIGHTGUN
Light gun.

DI8DEVTYPESCREENPTR_LIGHTPEN
Light pen.

DI8DEVTYPESCREENPTR_TOUCH
Touch screen.

DI8DEVTYPE_SUPPLEMENTAL
Specialized device with functionality unsuitable for the main control of an application, such as pedals used with a wheel. The following subtypes are defined.

DI8DEVTYPESUPPLEMENTAL_2NDHANDCONTROLLER
Secondary handheld controller.

DI8DEVTYPESUPPLEMENTAL_COMBINEDPEDALS
Device whose primary function is to report acceleration and brake pedal values from a single axis.

DI8DEVTYPESUPPLEMENTAL_DUALPEDALS
Device whose primary function is to report acceleration and brake pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_HANDTRACKER
Device that tracks hand movement.

DI8DEVTYPESUPPLEMENTAL_HEADTRACKER
Device that tracks head movement.

DI8DEVTYPESUPPLEMENTAL_RUDDERPEDALS
Device with rudder pedals.

DI8DEVTYPESUPPLEMENTAL_SHIFTER
Device that reports gear selection from an axis.

DI8DEVTYPESUPPLEMENTAL_SHIFTSTICKGATE
Device that reports gear selection from button states.

DI8DEVTYPESUPPLEMENTAL_SPLITTHROTTLE
Device whose primary function is to report at least two throttle values. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_THREEPEDALS
Device whose primary function is to report acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_THROTTLE
Device whose primary function is to report a single throttle value. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_UNKNOWN
Unknown subtype.

Versions of DirectInput earlier than Microsoft DirectX® 8.0 have a somewhat different scheme of device types and subtypes. See the DIDEVTYPExxx defines in Dinput.h.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIDEVICEOBJECTDATA

Contains buffered device information. This structure is used with the **IDirectInputDevice8::GetDeviceData** and the **IDirectInputDevice8::SendDeviceData** methods.

```
typedef struct DIDEVICEOBJECTDATA {
    DWORD    dwOfs;
    DWORD    dwData;
    DWORD    dwTimeStamp;
    DWORD    dwSequence;
    UINT_PTR uAppData;
} DIDEVICEOBJECTDATA, *LPDIDEVICEOBJECTDATA;

typedef const DIDEVICEOBJECTDATA *LPCDIDEVICEOBJECTDATA;
```

Members

dwOfs
For **GetDeviceData**, the offset into the current data format of the object whose data is being reported; that is, the location in which the **dwData** would have been stored if the data had been obtained by a call to the **IDirectInputDevice8::GetDeviceState** method. If the device is accessed as a mouse, keyboard, or joystick, the **dwOfs** member is one of the mouse device

constants, keyboard device constants, or joystick device constants. If a custom data format has been set, it is an offset relative to the custom data format.

This member can be ignored if Action Mapping is being used. Instead, retrieve the action value from **uAppData**.

For **SendDeviceData**, the instance ID of the object to which the data is being sent, as obtained from the **dwType** member of a **DIDEVICEOBJECTINSTANCE** structure.

dwData

Data obtained from or sent to the device.

For axis input, if the device is in relative axis mode, the relative axis motion is reported. If the device is in absolute axis mode, the absolute axis coordinate is reported.

For button input, only the low byte of **dwData** is significant. The high bit of the low byte is set if the button was pressed; it is clear if the button was released.

dwTimeStamp

System time at which the input event was generated, in milliseconds. This value wraps around approximately every 50 days. See Remarks.

When the structure is used with the **SendDeviceData** method, this member must be 0.

dwSequence

Microsoft® DirectInput® sequence number for this event. All input events are assigned an increasing sequence number. This enables events from different devices to be sorted chronologically. Because this value can wrap around, care must be taken when comparing two sequence numbers. The **DISEQUENCE_COMPARE** macro can be used to perform this comparison safely.

When the structure is used with the **SendDeviceData** method, this member must be 0.

uAppData

Application-defined action value assigned to this object in the last call to **IDirectInputDevice8::SetActionMap**. This is the value in the **uAppData** member of the **DIACTION** structure associated with the object. Ignore this value if you are not using action mapping.

When the structure is used with the **SendDeviceData** method, this member must be zero.

Remarks

The system time returned in **dwTimeStamp** comes from the same clock used by the Microsoft® Win32® **GetTickCount** or **timeGetTime** functions, but it produces potentially more precise values. For example, on Microsoft® Windows® 95, the **GetTickCount** timer is updated only every 55 milliseconds, but the **dwTimeStamp** value is accurate to within 1 millisecond. Therefore, if you call **GetTickCount** and it returns *n*, and you then receive an event with a timestamp of *n* + *n1*, you cannot

assume that the event took place exactly *n1* milliseconds after the call to **GetTickCount**.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIDEVICEOBJECTINSTANCE

Describes a device object instance. This structure is used with the **IDirectInputDevice8::EnumObjects** method to provide the **DIEnumDeviceObjectsCallback** callback function with information about a particular object associated with a device, such as an axis or button. It is also used with the **IDirectInputDevice8::GetObjectInfo** method to retrieve information about a device object.

```
typedef struct DIDEVICEOBJECTINSTANCE {
    DWORD dwSize;
    GUID  guidType;
    DWORD dwOfs;
    DWORD dwType;
    DWORD dwFlags;
    TCHAR tszName[MAX_PATH];
    DWORD dwFFMaxForce;
    DWORD dwFFForceResolution;
    WORD  wCollectionNumber;
    WORD  wDesignatorIndex;
    WORD  wUsagePage;
    WORD  wUsage;
    DWORD dwDimension;
    WORD  wExponent;
    WORD  wReportId;
} DIDEVICEOBJECTINSTANCE, *LPDIDEVICEOBJECTINSTANCE;

typedef const DIDEVICEOBJECTINSTANCE
*LPCDIDEVICEOBJECTINSTANCE;
```

Members

dwSize
Size of the structure, in bytes. During enumeration, the application can inspect this value to determine how many members of the structure are valid. When the structure is passed to the **IDirectInputDevice8::GetObjectInfo** method, this member must be initialized to **sizeof(DIDEVICEOBJECTINSTANCE)**.

guidType

Unique identifier that indicates the object type. This member is optional. If present, it can be one of the following values:

GUID_XAxis
The horizontal axis. For example, it can represent the left-right motion of a mouse.

GUID_YAxis
The vertical axis. For example, it can represent the forward-backward motion of a mouse.

GUID_ZAxis
The z-axis. For example, it can represent the rotation of a mouse wheel or the movement of a joystick throttle control.

GUID_RxAxis
Rotation around the x-axis.

GUID_RyAxis
Rotation around the y-axis.

GUID_RzAxis
Rotation around the z-axis (often a rudder control).

GUID_Slider
A slider axis.

GUID_Button
A mouse button.

GUID_Key
A keyboard key.

GUID_POV
A point-of-view indicator.

GUID_Unknown
Unknown.

Other object types might be defined in the future.

dwOfs

Offset within the data format at which data is reported for this object. This value can be used to identify the object in method calls and structures that accept the DIPH_BYOFFSET flag. See Remarks.

dwType

Device type that describes the object. It is a combination of DIDFT_* flags that describe the object type (axis, button, and so on) and contains the object instance number in the middle 16 bits. Use the **DIDFT_GETINSTANCE** macro to extract the object instance number. For the DIDFT_* flags, see **IDirectInputDevice8::EnumObjects**.

dwFlags

Flags describing other attributes of the data format. This value can be one of the following:

DIDOI_ASPECTACCEL

The object reports acceleration information.

DIDOI_ASPECTFORCE
The object reports force information.

DIDOI_ASPECTMASK
The bits that are used to report aspect information. An object can represent at most one aspect.

DIDOI_ASPECTPOSITION
The object reports position information.

DIDOI_ASPECTVELOCITY
The object reports velocity information.

DIDOI_FFACTUATOR
The object can have force-feedback effects applied to it.

DIDOI_FFEFFECTTRIGGER
The object can trigger playback of force-feedback effects.

DIDOI_GUIDISUSAGE
The **pguid** member of the **DIOBJECTDATAFORMAT** structure contains the desired usage page and usage in a packed **DWORD**. See **DIMAKEUSAGEDWORD**.

DIDOI_POLLED
The object does not return data until the **IDirectInputDevice8::Poll** method is called.

tszName[MAX_PATH]
Name of the object; for example, "X-Axis" or "Right Shift."

dwFFMaxForce
The magnitude of the maximum force that can be created by the actuator associated with this object. Force is expressed in newtons and measured in relation to where the hand would be during normal operation of the device.

dwFFForceResolution
The force resolution of the actuator associated with this object. The returned value represents the number of gradations, or subdivisions, of the maximum force that can be expressed by the force-feedback system from 0 (no force) to maximum force.

wCollectionNumber
The Human Interface Device (HID) link collection to which the object belongs.

wDesignatorIndex
An index that refers to a designator in the HID physical descriptor. This number can be passed to functions in the HID parsing library (Hidpi.h) to obtain additional information about the device object.

wUsagePage
The Human Interface Device (HID) usage page associated with the object, if known. HIDs always report a usage page. Non-HID devices can optionally report a usage page; if they do not, the value of this member is 0.

wUsage

The HID usage associated with the object, if known. HIDs always report a usage. Non-HIDs can optionally report a usage; if they do not, the value of this member is 0.

dwDimension
A Human Interface Device (HID)code for the dimensional units in which the object's value is reported, if known, or 0 if not known.

wExponent
The exponent to associate with the dimension, if known. Dimensional units are always integral, so an exponent might be needed to convert them to nonintegral types.

wReportId
Reserved.

Remarks

If a data format has been set for the device, the value in **dwOfs** is the offset of the object's data within that format. If no format has been set, the value is the offset within the raw data returned by the device. Applications should not use this value unless a data format has been set.

Applications can use the **wUsagePage** and **wUsage** members to obtain additional information about how the object was designed to be used. For example, if **wUsagePage** has the value 0x02 (vehicle controls) and **wUsage** has the value 0xB9 (elevator trim), the object was designed to be the elevator trim control on a flight stick. A flight simulator application can use this information to provide more reasonable defaults for objects on the device. HID usage codes are determined by the USB standards committee.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIEFFECT

Used by the **IDirectInputDevice8::CreateEffect** method to initialize a new **IDirectInputEffect** object. It is also used by the **IDirectInputEffect::SetParameters** and **IDirectInputEffect::GetParameters** methods.

```
typedef struct DIEFFECT {
    DWORD   dwSize;
    DWORD   dwFlags;
    DWORD   dwDuration;
    DWORD   dwSamplePeriod;
```

```
    DWORD   dwGain;
    DWORD   dwTriggerButton;
    DWORD   dwTriggerRepeatInterval;
    DWORD   cAxes;
    LPDWORD rgdwAxes;
    LPLONG  rglDirection;
    LPDIENVELOPE lpEnvelope;
    DWORD   cbTypeSpecificParams;
    LPVOID  lpvTypeSpecificParams;
    DWORD   dwStartDelay;
} DIEFFECT, *LPDIEFFECT;

typedef const DIEFFECT *LPCDIEFFECT;
```

Members

dwSize

Specifies the size, in bytes, of the structure. This member must be initialized before the structure is used.

dwFlags

Flags associated with the effect. This value can be a combination of one or more of the following values:

DIEFF_CARTESIAN
The values of **rglDirection** are to be interpreted as Cartesian coordinates.

DIEFF_OBJECTIDS
The values of **dwTriggerButton** and **rgdwAxes** are object identifiers as obtained by **IDirectInputDevice8::EnumObjects**.

DIEFF_OBJECTOFFSETS
The values of **dwTriggerButton** and **rgdwAxes** are data format offsets.

DIEFF_POLAR
The values of **rglDirection** are to be interpreted as polar coordinates.

DIEFF_SPHERICAL
The values of **rglDirection** are to be interpreted as spherical coordinates.

dwDuration

The total duration of the effect, in microseconds. If this value is INFINITE, the effect has infinite duration. If an envelope has been applied to the effect, the attack is applied, followed by an infinite sustain.

dwSamplePeriod

The period at which the device should play back the effect, in microseconds. A value of 0 indicates that the default playback sample rate should be used.

If the device is not capable of playing back the effect at the specified rate, it chooses the supported rate that is closest to the requested value.

Setting a custom **dwSamplePeriod** can be used for special effects. For example, playing a sine wave at an artificially large sample period results in a rougher texture.

dwGain

The gain to be applied to the effect, in the range from 0 through 10,000. The gain is a scaling factor applied to all magnitudes of the effect and its envelope.

dwTriggerButton

The identifier or offset of the button to be used to trigger playback of the effect. The flags DIEFF_OBJECTIDS and DIEFF_OBJECTOFFSETS determine the semantics of the value. If this member is set to DIEB_NOTRIGGER, no trigger button is associated with the effect.

dwTriggerRepeatInterval

The interval, in microseconds, between the end of one playback and the start of the next when the effect is triggered by a button press and the button is held down. Setting this value to INFINITE suppresses repetition.

Support for trigger repeat for an effect is indicated by the presence of the DIEP_TRIGGERREPEATINTERVAL flag in the **dwStaticParams** member of the **DIEFFECTINFO** structure.

cAxes

Number of axes involved in the effect. This member must be filled in by the caller if changing or setting the axis list or the direction list.

The number of axes for an effect cannot be changed once it has been set.

rgdwAxes

Pointer to a **DWORD** array (of **cAxes** elements) containing identifiers or offsets identifying the axes to which the effect is to be applied. The flags DIEFF_OBJECTIDS and DIEFF_OBJECTOFFSETS determine the semantics of the values in the array.

The list of axes associated with an effect cannot be changed once it has been set.

No more than 32 axes can be associated with a single effect.

rglDirection

Pointer to a **LONG** array (of **cAxes** elements) containing either Cartesian coordinates, polar coordinates, or spherical coordinates. The flags DIEFF_CARTESIAN, DIEFF_POLAR, and DIEFF_SPHERICAL determine the semantics of the values in the array.

If Cartesian, each value in **rglDirection** is associated with the corresponding axis in **rgdwAxes**.

If polar, the angle is measured in hundredths of degrees from the (0, –1) direction, rotated in the direction of (1, 0). This usually means that north is away from the user, and east is to the user's right. The last element is not used.

If spherical, the first angle is measured in hundredths of a degree from the (1, 0) direction, rotated in the direction of (0, 1). The second angle (if the number of axes is three or more) is measured in hundredths of a degree toward (0, 0, 1). The third angle (if the number of axes is four or more) is measured in hundredths of a degree toward (0, 0, 0, 1), and so on. The last element is not used.

Note

The **rglDirection** array must contain **cAxes** entries, even if polar or spherical coordinates are given. In these cases, the last element in the **rglDirection** array is reserved for future use and must be 0.

lpEnvelope
Optional pointer to a **DIENVELOPE** structure that describes the envelope to be used by this effect. Not all effect types use envelopes. If no envelope is to be applied, the member should be set to NULL.

cbTypeSpecificParams
Number of bytes of additional type-specific parameters for the corresponding effect type.

lpvTypeSpecificParams
Pointer to type-specific parameters, or NULL if there are no type-specific parameters.

If the effect is of type DIEFT_CONDITION, this member contains a pointer to an array of **DICONDITION** structures that define the parameters for the condition. A single structure may be used, in which case the condition is applied in the direction specified in the **rglDirection** array. Otherwise, there must be one structure for each axis, in the same order as the axes in **rgdwAxes** array. If a structure is supplied for each axis, the effect should not be rotated; you should use the following values in the **rglDirection** array:

- DIEFF_SPHERICAL: 0, 0, ...
- DIEFF_POLAR: 9000, 0, ...
- DIEFF_CARTESIAN: 1, 0, ...

If the effect is of type DIEFT_CUSTOMFORCE, this member contains a pointer to a **DICUSTOMFORCE** structure that defines the parameters for the custom force.

If the effect is of type DIEFT_PERIODIC, this member contains a pointer to a **DIPERIODIC** structure that defines the parameters for the effect.

If the effect is of type DIEFT_CONSTANTFORCE, this member contains a pointer to a **DICONSTANTFORCE** structure that defines the parameters for the constant force.

If the effect is of type DIEFT_RAMPFORCE, this member contains a pointer to a **DIRAMPFORCE** structure that defines the parameters for the ramp force.

dwStartDelay
Time (in microseconds) that the device should wait after a **IDirectInputEffect::Start** call before playing the effect. If this value is 0, effect playback begins immediately. This member is not present in versions prior to Microsoft® DirectX® 7.0.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for

Windows 95.
Header: Declared in Dinput.h.

DIEFFECTINFO

Used by the **IDirectInputDevice8::EnumEffects** and **IDirectInputDevice8::GetEffectInfo** methods to return information about a particular effect supported by a device.

```
typedef struct DIEFFECTINFO {
    DWORD dwSize;
    GUID  guid;
    DWORD dwEffType;
    DWORD dwStaticParams;
    DWORD dwDynamicParams;
    TCHAR tszName[MAX_PATH];
} DIEFFECTINFO, *LPDIEFFECTINFO;

typedef const DIEFFECTINFO  *LPCDIEFFECTINFO;
```

Members

dwSize
Size of the structure in bytes. During enumeration, the application can inspect this value to determine how many members of the structure are valid. This member must be initialized before the structure is passed to the **IDirectInputDevice8::GetEffectInfo** method.

guid
Identifier of the effect.

dwEffType
Zero or more of the following values:

DIEFT_ALL
Valid only for **IDirectInputDevice8::EnumEffects.** Enumerate all effects, regardless of type. This flag cannot be combined with any of the other flags.

DIEFT_CONDITION
The effect represents a condition. When creating or modifying a condition, the **lpvTypeSpecificParams** member of the **DIEFFECT** structure must point to an array of **DICONDITION** structures (one per axis), and the **cbTypeSpecificParams** member must be set to **cAxis * sizeof(DICONDITION)**.

Not all devices support all the parameters of conditions. Check the effect capability flags to determine which capabilities are available.

The flag can be passed to **IDirectInputDevice8::EnumEffects** to restrict the enumeration to conditions.

DIEFT_CONSTANTFORCE

The effect represents a constant-force effect. When creating or modifying a constant-force effect, the **lpvTypeSpecificParams** member of the **DIEFFECT** must point to a **DICONSTANTFORCE** structure, and the **cbTypeSpecificParams** member must be set to **sizeof(DICONSTANTFORCE)**.

The flag can be passed to **IDirectInputDevice8::EnumEffects** to restrict the enumeration to constant-force effects.

DIEFT_CUSTOMFORCE

The effect represents a custom-force effect. When creating or modifying a custom-force effect, the **lpvTypeSpecificParams** member of the **DIEFFECT** structure must point to a **DICUSTOMFORCE** structure, and the **cbTypeSpecificParams** member must be set to **sizeof(DICUSTOMFORCE)**.

The flag can be passed to **IDirectInputDevice8::EnumEffects** to restrict the enumeration to custom-force effects.

DIEFT_DEADBAND

The effect generator for this condition effect supports the **lDeadBand** parameter.

DIEFT_FFATTACK

The effect generator for this effect supports the attack envelope parameter. If the effect generator does not support attack, the attack level and attack time parameters of the **DIENVELOPE** structure are ignored by the effect.

If neither DIEFT_FFATTACK nor DIEFT_FFFADE is set, the effect does not support an envelope, and any provided envelope is ignored.

DIEFT_FFFADE

The effect generator for this effect supports the fade parameter. If the effect generator does not support fade, the fade level and fade time parameters of the **DIENVELOPE** structure are ignored by the effect.

If neither DIEFT_FFATTACK nor DIEFT_FFFADE is set, the effect does not support an envelope, and any provided envelope is ignored.

DIEFT_HARDWARE

The effect represents a hardware-specific effect. For additional information on using a hardware-specific effect, consult the hardware documentation.

The flag can be passed to the **IDirectInputDevice8::EnumEffects** method to restrict the enumeration to hardware-specific effects.

DIEFT_PERIODIC

The effect represents a periodic effect. When creating or modifying a periodic effect, the **lpvTypeSpecificParams** member of the **DIEFFECT** structure must point to a **DIPERIODIC** structure, and the **cbTypeSpecificParams** member must be set to **sizeof(DIPERIODIC)**.

The flag can be passed to **IDirectInputDevice8::EnumEffects** to restrict the enumeration to periodic effects.

DIEFT_POSNEGCOEFFICIENTS

The effect generator for this effect supports two coefficient values for conditions, one for the positive displacement of the axis and one for the negative displacement of the axis. If the device does not support both coefficients, the negative coefficient in the **DICONDITION** structure is ignored, and the positive coefficient is used in both directions.

DIEFT_POSNEGSATURATION
The effect generator for this effect supports a maximum saturation for both positive and negative force output. If the device does not support both saturation values, the negative saturation in the **DICONDITION** structure is ignored, and the positive saturation is used in both directions.

DIEFT_RAMPFORCE
The effect represents a ramp-force effect. When creating or modifying a ramp-force effect, the **lpvTypeSpecificParams** member of the **DIEFFECT** structure must point to a **DIRAMPFORCE** structure, and the **cbTypeSpecificParams** member must be set to **sizeof(DIRAMPFORCE**).

The flag can be passed to **IDirectInputDevice8::EnumEffects** to restrict the enumeration to ramp-force effects.

DIEFT_SATURATION
The effect generator for this effect supports the saturation of condition effects. If the effect generator does not support saturation, the force generated by a condition is limited only by the maximum force that the device can generate.

DIEFT_STARTDELAY
The effect has a delay before it plays rather than beginning immediately. This can be used to offset effects so that they play in sequence.

dwStaticParams
Zero or more DIEP_* values describing the parameters supported by the effect. For example, if DIEP_ENVELOPE is set, the effect supports an envelope. For a list of possible values, see **IDirectInputEffect::GetParameters**.

It is not an error for an application to attempt to use effect parameters that are not supported by the device. The unsupported parameters are ignored.

This information is provided to enable the application to tailor its use of force feedback to the capabilities of the specific device.

dwDynamicParams
Zero or more DIEP_* values denoting parameters of the effect that can be modified while the effect is playing. For a list of possible values, see **IDirectInputEffect::GetParameters**.

If an application attempts to change a parameter while the effect is playing and the driver does not support modifying that effect dynamically, the driver is permitted to stop the effect, update the parameters, then restart it. For more information, see **IDirectInputEffect::SetParameters**.

tszName[MAX_PATH]
Name of the effect; for example, "Sawtooth up" or "Constant force".

Remarks

Use the **DIEFT_GETTYPE** macro to extract the effect type from the **dwEffType** flags.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIEFFESCAPE

Used by the **IDirectInputDevice8::Escape** and **IDirectInputEffect::Escape** methods to pass hardware-specific data directly to the device driver.

```
typedef struct DIEFFESCAPE {
    DWORD  dwSize;
    DWORD  dwCommand;
    LPVOID lpvInBuffer;
    DWORD  cbInBuffer;
    LPVOID lpvOutBuffer;
    DWORD  cbOutBuffer;
} DIEFFESCAPE, *LPDIEFFESCAPE;
```

Members

dwSize
Size of the structure in bytes. This member must be initialized before the structure is used.

dwCommand
Driver-specific command number. Consult the driver documentation for a list of valid commands.

lpvInBuffer
Buffer containing the data required to perform the operation.

cbInBuffer
Size, in bytes, of the **lpvInBuffer** buffer.

lpvOutBuffer
Buffer in which the operation's output data is returned.

cbOutBuffer
On entry, the size in bytes of the **lpvOutBuffer** buffer. On exit, the number of bytes actually produced by the command.

Remarks

Because each driver implements different escapes, it is the application's responsibility to ensure that it is talking to the correct driver by comparing the **guidFFDriver** member in the **DIDEVICEINSTANCE** structure against the value the application is expecting.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIENVELOPE

Used by the **DIEFFECT** structure to specify the optional envelope parameters for an effect. The sustain level for the envelope is represented by the **dwMagnitude** member of the **DIPERIODIC** structure and the **lMagnitude** member of the **DICONSTANTFORCE** structure. The sustain time is represented by **dwDuration** member of the **DIEFFECT** structure.

```
typedef struct DIENVELOPE {
    DWORD dwSize;
    DWORD dwAttackLevel;
    DWORD dwAttackTime;
    DWORD dwFadeLevel;
    DWORD dwFadeTime;
} DIENVELOPE, *LPDIENVELOPE;

typedef const DIENVELOPE *LPCDIENVELOPE;
```

Members

dwSize
Size, in bytes, of the structure. This member must be initialized before the structure is used.

dwAttackLevel
Amplitude for the start of the envelope, relative to the baseline, in the range from 0 through 10,000. If the effect's type-specific data does not specify a baseline, the amplitude is relative to 0.

dwAttackTime
The time, in microseconds, to reach the sustain level.

dwFadeLevel
Amplitude for the end of the envelope, relative to the baseline, in the range from 0 through 10,000. If the effect's type-specific data does not specify a baseline, the amplitude is relative to 0.

dwFadeTime
The time, in microseconds, to reach the fade level.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIFILEEFFECT

Describes data for a force-feedback effect stored in a file. It is used in conjunction with the **IDirectInputDevice8::EnumEffectsInFile** and **IDirectInputDevice8::WriteEffectToFile** methods.

```
typedef struct DIFILEEFFECT{
    DWORD       dwSize;
    GUID        GuidEffect;
    LPCDIEFFECT lpDiEffect;
    CHAR        szFriendlyName[MAX_PATH];
}DIFILEEFFECT, *LPDIFILEEFFECT;
```

Members

dwSize
Size, in bytes, of the structure. This member must be initialized before the structure is used.

GuidEffect
Unique identifier of the effect type. This can be one of the standard GUIDs defined in Dinput.h, such as GUID_ConstantForce, or one created by the designer.

lpDiEffect
Pointer to a **DIEFFECT** structure containing information about the effect.

szFriendlyName
Name of the effect.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIJOYSTATE

Describes the state of a joystick device. This structure is used with the **IDirectInputDevice8::GetDeviceState** method.

```
typedef struct DIJOYSTATE {
    LONG    lX;
    LONG    lY;
    LONG    lZ;
    LONG    lRx;
    LONG    lRy;
    LONG    lRz;
    LONG    rglSlider[2];
    DWORD   rgdwPOV[4];
    BYTE    rgbButtons[32];
} DIJOYSTATE, *LPDIJOYSTATE;
```

Members

lX
X-axis, usually the left-right movement of a stick.

lY
Y-axis, usually the forward-backward movement of a stick.

lZ
Z-axis, often the throttle control. If the joystick does not have this axis, the value is 0.

lRx
X-axis rotation. If the joystick does not have this axis, the value is 0.

lRy
Y-axis rotation. If the joystick does not have this axis, the value is 0.

lRz
Z-axis rotation (often called the rudder). If the joystick does not have this axis, the value is 0.

rglSlider[2]
Two additional axes, formerly called the u-axis and v-axis, whose semantics depend on the joystick. Use the **IDirectInputDevice8::GetObjectInfo** method to obtain semantic information about these values.

rgdwPOV[4]
Direction controllers, such as point-of-view hats. The position is indicated in hundredths of a degree clockwise from north (away from the user). The center position is normally reported as –1; but see Remarks. For indicators that have only five positions, the value for a controller is –1, 0, 9,000, 18,000, or 27,000.

rgbButtons[32]
Array of buttons. The high-order bit of the byte is set if the corresponding button is down, and clear if the button is up or does not exist.

Remarks

You must prepare the device for joystick-style access by calling the **IDirectInputDevice8::SetDataFormat** method, passing the *c_dfDIJoystick* global data format variable.

If an axis is in relative mode, the appropriate member contains the change in position. If it is in absolute mode, the member contains the absolute axis position.

Some drivers report the centered position of the POV indicator as 65,535. Determine whether the indicator is centered as follows:

```
BOOL POVCentered = (LOWORD(dwPOV) == 0xFFFF);
```

Note

Under Microsoft® DirectX® 7, sliders on some joysticks could be assigned to the Z axis, with subsequent code retrieving data from that member. Using DirectX 8, those same sliders will be assigned to the **rglSlider** array. This should be taken into account when porting applications to DirectX 8. Make any necessary alterations to ensure that slider data is retrieved from the rglSlider array.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIJOYSTATE2

DIJOYSTATE2

Describes the state of a joystick device with extended capabilities. This structure is used with the **IDirectInputDevice8::GetDeviceState** method.

```
typedef struct DIJOYSTATE2 {
    LONG    lX;
    LONG    lY;
    LONG    lZ;
    LONG    lRx;
    LONG    lRy;
    LONG    lRz;
    LONG    rglSlider[2];
    DWORD   rgdwPOV[4];
    BYTE    rgbButtons[128];
    LONG    lVX;
```

```
    LONG    lVY;
    LONG    lVZ;
    LONG    lVRx;
    LONG    lVRy;
    LONG    lVRz;
    LONG    rglVSlider[2];
    LONG    lAX;
    LONG    lAY;
    LONG    lAZ;
    LONG    lARx;
    LONG    lARy;
    LONG    lARz;
    LONG    rglASlider[2];
    LONG    lFX;
    LONG    lFY;
    LONG    lFZ;
    LONG    lFRx;
    LONG    lFRy;
    LONG    lFRz;
    LONG    rglFSlider[2];
} DIJOYSTATE2, *LPDIJOYSTATE2;
```

Members

lX

X-axis, usually the left-right movement of a stick.

lY

Y-axis, usually the forward-backward movement of a stick.

lZ

Z-axis, often the throttle control. If the joystick does not have this axis, the value is 0.

lRx

X-axis rotation. If the joystick does not have this axis, the value is 0.

lRy

Y-axis rotation. If the joystick does not have this axis, the value is 0.

lRz

Z-axis rotation (often called the rudder). If the joystick does not have this axis, the value is 0.

rglSlider[2]

Two additional axis values (formerly called the u-axis and v-axis) whose semantics depend on the joystick. Use the **IDirectInputDevice8::GetObjectInfo** method to obtain semantic information about these values.

rgdwPOV[4]

Direction controllers, such as point-of-view hats. The position is indicated in hundredths of a degree clockwise from north (away from the user). The center position is normally reported as –1; but see Remarks. For indicators that have only five positions, the value for a controller is –1, 0, 9,000, 18,000, or 27,000.

rgbButtons[128]
Array of buttons. The high-order bit of the byte is set if the corresponding button is down, and clear if the button is up or does not exist.

lVX
X-axis velocity.

lVY
Y-axis velocity.

lVZ
Z-axis velocity.

lVRx
X-axis angular velocity.

lVRy
Y-axis angular velocity.

lVRz
Z-axis angular velocity.

rglVSlider[2]
Extra axis velocities.

lAX
X-axis acceleration.

lAY
Y-axis acceleration.

lAZ
Z-axis acceleration.

lARx
X-axis angular acceleration.

lARy
Y-axis angular acceleration.

lARz
Z-axis angular acceleration.

rglASlider[2]
Extra axis accelerations.

lFX
X-axis force.

lFY
Y-axis force.

lFZ
Z-axis force.

lFRx
X-axis torque.

lFRy
Y-axis torque.

lFRz
Z-axis torque.

rglFSlider[2]
Extra axis forces.

Remarks

You must prepare the device for access to a joystick with extended capabilities by calling the **IDirectInputDevice8::SetDataFormat** method, passing the *c_dfDIJoystick2* global data format variable.

If an axis is in relative mode, the appropriate member contains the change in position. If it is in absolute mode, the member contains the absolute axis position.

Some drivers report the centered position of the POV indicator as 65,535. Determine whether the indicator is centered as follows:

```
BOOL POVCentered = (LOWORD(dwPOV) == 0xFFFF);
```

Note

Under Microsoft® DirectX® 7, sliders on some joysticks could be assigned to the Z axis, with subsequent code retrieving data from that member. Using DirectX 8, those same sliders will be assigned to the **rglSlider** array. This should be taken into account when porting applications to DirectX 8. Make any necessary alterations to ensure that slider data is retrieved from the rglSlider array.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIJOYSTATE

DIMOUSESTATE

Describes the state of a mouse device that has up to four buttons, or another device that is being accessed as if it were a mouse device. This structure is used with the **IDirectInputDevice8::GetDeviceState** method.

```
typedef struct DIMOUSESTATE {
  LONG lX;
```

```
    LONG lY;
    LONG lZ;
    BYTE rgbButtons[4];
} DIMOUSESTATE, *LPDIMOUSESTATE;
```

Members

lX
X-axis.

lY
Y-axis.

lZ
Z-axis, typically a wheel. If the mouse does not have a z-axis, the value is 0.

rgbButtons
Array of buttons. The high-order bit of the byte is set if the corresponding button is down.

Remarks

You must prepare the device for mouse-style access by calling the **IDirectInputDevice8::SetDataFormat** method, passing the *c_dfDIMouse* global data format variable.

The mouse is a relative-axis device, so the absolute axis positions for mouse axes are simply accumulated relative motion. Therefore, the value of the absolute axis position is not meaningful except in comparison with other absolute axis positions.

If an axis is in relative mode, the appropriate member contains the change in position. If it is in absolute mode, the member contains the absolute axis position.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIMOUSESTATE2

DIMOUSESTATE2

Describes the state of a mouse device that has up to eight buttons, or another device that is being accessed as if it were a mouse device. This structure is used with the **IDirectInputDevice8::GetDeviceState** method.

```
typedef struct DIMOUSESTATE {
    LONG lX;
    LONG lY;
    LONG lZ;
    BYTE rgbButtons[8];
} DIMOUSESTATE, *LPDIMOUSESTATE;
```

Members

lX
X-axis.

lY
Y-axis.

lZ
Z-axis, typically a wheel. If the mouse does not have a z-axis, the value is 0.

rgbButtons
Array of buttons. The high-order bit of the byte is set if the corresponding button is down.

Remarks

You must prepare the device for mouse-style access by calling the **IDirectInputDevice8::SetDataFormat** method, passing the *c_dfDIMouse2* global data format variable.

The mouse is a relative-axis device, so the absolute axis positions for mouse axes are simply accumulated relative motion. Therefore, the value of the absolute axis position is not meaningful except in comparison with other absolute axis positions.

If an axis is in relative mode, the appropriate member contains the change in position. If it is in absolute mode, the member contains the absolute axis position.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

DIMOUSESTATE

DIOBJECTDATAFORMAT

Describes a device object's data format for use with the **IDirectInputDevice8::SetDataFormat** method.

```
typedef struct DIOBJECTDATAFORMAT {
    const GUID * pguid;
    DWORD       dwOfs;
    DWORD       dwType;
    DWORD       dwFlags;
} DIOBJECTDATAFORMAT, *LPDIOBJECTDATAFORMAT;

typedef const DIOBJECTDATAFORMAT *LPCDIOBJECTDATAFORMAT;
```

Members

pguid
Unique identifier for the axis, button, or other input source. When requesting a data format, making this member NULL indicates that any type of object is permissible.

dwOfs
Offset within the data packet where the data for the input source is stored. This value must be a multiple of 4 for **DWORD** size data, such as axes. It can be byte-aligned for buttons.

dwType
Device type that describes the object. It is a combination of the following flags describing the object type (axis, button, and so forth) and containing the object-instance number in the middle 16 bits. When requesting a data format, the instance portion must be set to DIDFT_ANYINSTANCE to indicate that any instance is permissible, or to **DIDFT_MAKEINSTANCE**(*n*) to restrict the request to instance *n*. See the examples under Remarks.

DIDFT_ABSAXIS
The object selected by the **SetDataFormat** method must be an absolute axis.

DIDFT_AXIS
The object selected by the **SetDataFormat** method must be an absolute or relative axis.

DIDFT_BUTTON
The object selected by the **SetDataFormat** method must be a push button or a toggle button.

DIDFT_FFACTUATOR
The object selected by the **SetDataFormat** method must contain a force-feedback actuator; in other words, it must be possible to apply forces to the object.

DIDFT_FFEFFECTTRIGGER
The object selected by the **SetDataFormat** method must be a valid force-feedback effect trigger.

DIDFT_POV
The object selected by the **SetDataFormat** method must be a point-of-view controller.

DIDFT_PSHBUTTON
The object selected by the **SetDataFormat** method must be a push button.

DIDFT_RELAXIS
The object selected by **SetDataFormat** must be a relative axis.

DIDFT_TGLBUTTON
The object selected by **SetDataFormat** must be a toggle button.

DIDFT_VENDORDEFINED
The object selected by **SetDataFormat** must be of a type defined by the manufacturer.

dwFlags
Zero or more of the following values:

DIDOI_ASPECTACCEL
The object selected by **SetDataFormat** must report acceleration information.

DIDOI_ASPECTFORCE
The object selected by **SetDataFormat** must report force information.

DIDOI_ASPECTPOSITION
The object selected by **SetDataFormat** must report position information.

DIDOI_ASPECTVELOCITY
The object selected by **SetDataFormat** must report velocity information.

Remarks

A data format is made up of several **DIOBJECTDATAFORMAT** structures, one for each object (axis, button, and so on). An array of these structures is contained in the **DIDATAFORMAT** structure that is passed to **IDirectInputDevice8::SetDataFormat**. An application typically does not need to create an array of **DIOBJECTDATAFORMAT** structures; rather, it can use one of the predefined data formats, *c_dfDIMouse*, *c_dfDIMouse2*, *c_dfDIKeyboard*, *c_dfDIJoystick*, or *c_dfDIJoystick2*, which have predefined settings for **DIOBJECTDATAFORMAT**.

The following object data format specifies that Microsoft® DirectInput® should choose the first available axis and report its value in the **DWORD** at offset 4 in the device data.

```
DIOBJECTDATAFORMAT dfAnyAxis = {
    0,                  // Wildcard
    4,                  // Offset
    DIDFT_AXIS | DIDFT_ANYINSTANCE, // Any axis is okay.
    0,                  // Ignore aspect
};
```

The following object data format specifies that the x-axis of the device should be stored in the **DWORD** at offset 12 in the device data. If the device has more than one x-axis, the first available one should be selected.

```
DIOBJECTDATAFORMAT dfAnyXAxis = {
    &GUID_XAxis,                // Must be an X axis
    12,                         // Offset
    DIDFT_AXIS | DIDFT_ANYINSTANCE, // Any X axis is okay.
    0,                          // Ignore aspect
};
```

The following object data format specifies that DirectInput should choose the first available button and report its value in the high bit of the byte at offset 16 in the device data.

```
DIOBJECTDATAFORMAT dfAnyButton = {
    0,                          // Wildcard
    16,                         // Offset
    DIDFT_BUTTON | DIDFT_ANYINSTANCE, // Any button is okay.
    0,                          // Ignore aspect
};
```

The following object data format specifies that button 0 of the device should be reported as the high bit of the byte stored at offset 18 in the device data.

If the device does not have a button 0, the attempt to set this data format fails.

```
DIOBJECTDATAFORMAT dfButton0 = {
    0,                          // Wildcard
    18,                         // Offset
    DIDFT_BUTTON | DIDFT_MAKEINSTANCE(0), // Button zero
    0,                          // Ignore aspect
};
```

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIPERIODIC

Contains type-specific information for effects that are marked as **DIEFT_PERIODIC**.

The structure describes a periodic effect.

A pointer to a single **DIPERIODIC** structure for an effect is passed in the **lpvTypeSpecificParams** member of the **DIEFFECT** structure.

```
typedef struct DIPERIODIC {
    DWORD dwMagnitude;
    LONG  lOffset;
    DWORD dwPhase;
    DWORD dwPeriod;
} DIPERIODIC, *LPDIPERIODIC;

typedef const DIPERIODIC *LPCDIPERIODIC;
```

Members

dwMagnitude
Magnitude of the effect, in the range from 0 through 10,000. If an envelope is applied to this effect, the value represents the magnitude of the sustain. If no envelope is applied, the value represents the amplitude of the entire effect.

lOffset
Offset of the effect. The range of forces generated by the effect is **lOffset** minus **dwMagnitude** to **lOffset** plus **dwMagnitude**. The value of the **lOffset** member is also the baseline for any envelope that is applied to the effect.

dwPhase
Position in the cycle of the periodic effect at which playback begins, in the range from 0 through 35,999. See Remarks.

dwPeriod
Period of the effect, in microseconds.

Remarks

A device driver cannot provide support for all values in the **dwPhase** member. In this case, the value is rounded off to the nearest supported value.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIPROPCPOINTS

Sets calibration points for the current instance.

```
typedef struct DIPROPCPOINTS {
    DIPROPHEADER diph;
    DWORD  dwCPointsNum;
```

```
    CPOINT cp[MAXCPOINTSNUM];
} DIPROPCPOINTS, *LPDIPROPCPOINTS;

typedef const DIPROPCPOINTS *LPCDIPROPCPOINTS;
```

Members

diph

DIPROPHEADER structure that must be initialized as follows:

Member	Value
dwSize	**sizeof(DIPROPCPOINTS)**
dwHeaderSize	**sizeof(DIPROPHEADER)**
dwObj	If the **dwHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. If the **dwHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. For example, if the *c_dfDIMouse* data format is selected, it must be one of the DIMOFS_* values. If the **dwHow** member is DIPH_BYUSAGE, the device must be a Human Interface Device (HID). The device object will be identified by the HID usage page and usage values in packed form. This property can be applied only to individual device objects, so DIPH_DEVICE cannot be used.
dwHow	Specifies how the **dwObj** member should be interpreted. See the description of the **dwObj** member in this table for details.

dwCPointsNum

Number of stored calibration points. No more than 8 calibration points may be used.

cp[MAXCPOINTSNUM]

Array of calibration points.

DIPROPDWORD

Used to access **DWORD** properties.

```
typedef struct DIPROPDWORD {
    DIPROPHEADER diph;
    DWORD        dwData;
} DIPROPDWORD, *LPDIPROPDWORD;

typedef const DIPROPDWORD *LPCDIPROPDWORD;
```

Members

diph

DIPROPHEADER structure that must be initialized as follows:

Member	Value
dwSize	**sizeof(DIPROPDWORD)**
dwHeaderSize	**sizeof(DIPROPHEADER)**
dwObj	If the **dwHow** member is DIPH_DEVICE, this member must be 0. If the **dwHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. If the **dwHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. For example, if the *c_dfDIMouse* data format is selected, it must be one of the DIMOFS_* values. If the **dwHow** member is DIPH_BYUSAGE, the device must be a Human Interface Device (HID). The device object will be identified by the HID usage page and usage values in packed form.
dwHow	Specifies how the **dwObj** member should be interpreted. See the description of the **dwObj** member in this table for details.

dwData

Property-specific value being set or retrieved.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::GetProperty, **IDirectInputDevice8::SetProperty**

DIPROPGUIDANDPATH

Used to access properties whose values represent a GUID and a path.

```
typedef struct DIPROPGUIDANDPATH {
  DIPROPHEADER diph;
```

```
    GUID        guidClass;
    WCHAR       wszPath[MAX_PATH];
} DIPROPGUIDANDPATH, *LPDIPROPGUIDANDPATH;

typedef const DIPROPGUIDANDPATH *LPCDIPROPGUIDANDPATH;
```

Members

diph
DIPROPHEADER structure that must be initialized as follows:

Member	Value
dwSize	**sizeof(DIPROPGUIDANDPATH)**.
dwHeaderSize	**sizeof(DIPROPHEADER)**.
dwObj	For this structure, this member must be set to 0.
dwHow	Specifies how the **dwObj** member should be interpreted. For this structure, **dwHow** should be DIPH_DEVICE.

guidClass
Class GUID for the object.

wszPath
Returned path for the object. This is a Unicode string.

Remarks

The DIPROP_GUIDANDPATH property associated with the **DIPROPGUIDANDPATH** structure enables advanced applications to perform operations on a Human Interface Device (HID) that are not supported by Microsoft® DirectInput®.

The application calls the **IDirectInputDevice8::GetProperty** method with DIPROP_GUIDANDPATH as the *rguidProp* parameter. The class GUID of the device is returned in the **guidClass** member of the **DIPROPGUIDANDPATH** structure, and the device interface path is returned in the **wszPath** member. The application can then call the **CreateFile** function on this path to access the device directly.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 98.
Header: Declared in Dinput.h.

DIPROPHEADER

Serves as a header for all property structures.

```
typedef struct DIPROPHEADER {
    DWORD    dwSize;
    DWORD    dwHeaderSize;
    DWORD    dwObj;
    DWORD    dwHow;
} DIPROPHEADER, *LPDIPROPHEADER;

typedef const DIPROPHEADER *LPCDIPROPHEADER;
```

Members

dwSize
Size of the enclosing structure. This member must be initialized before the structure is used.

dwHeaderSize
Size of the **DIPROPHEADER** structure.

dwObj
Object for which the property is to be accessed. The value set for this member depends on the value specified in the **dwHow** member.

dwHow
Value that specifies how the **dwObj** member should be interpreted. This value can be one of the following:

Value	Meaning
DIPH_DEVICE	The **dwObj** member must be 0.
DIPH_BYOFFSET	The **dwObj** member is the offset into the current data format of the object whose property is being accessed.
DIPH_BYUSAGE	The **dwObj** member is the HID usage page and usage values in packed form.
DIPH_BYID	The **dwObj** member is the object type/instance identifier. This identifier is returned in the **dwType** member of the **DIDEVICEOBJECTINSTANCE** structure returned from a previous call to the **IDirectInputDevice8::EnumObjects** member.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

DIPROPPOINTER

Used in setting and retrieving data values of type UINT_PTR. This is typically a value associated with an application-specific action associated with a device object in an action map, but can also be a function pointer.

```
typedef struct DIPROPPOINTER {
    DIPROPHEADER diph;
    UINT_PTR     uData;
} DIPROPPOINTER, *LPDIPROPPOINTER;

typedef const DIPROPPOINTER *LPCDIPROPPOINTER;
```

Members

diph

DIPROPHEADER structure that must be initialized as follows:

Member	Value
dwSize	**sizeof(DIPROPPOINTER)**
dwHeaderSize	**sizeof(DIPROPHEADER)**
dwObj	If the **dwHow** member is DIPH_DEVICE, this member must be 0. If the **dwHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. If the **dwHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. For example, if the *c_dfDIMouse* data format is selected, it must be one of the DIMOFS_* values. If the **dwHow** member is DIPH_BYUSAGE, the device must be a Human Interface Device (HID). The device object will be identified by the HID usage page and usage values in packed form.
dwHow	Specifies how the **dwObj** member should be interpreted. See the description of the **dwObj** member in this table for details.

uData

Application-defined value of type UINT_PTR, which represents the uAppData value (from an action map) that is associated with a device control.

DIPROPRANGE

Contains information about the range of an object within a device. This structure is used with the DIPROP_RANGE flag set in the **IDirectInputDevice8::GetProperty** and **IDirectInputDevice8::SetProperty** methods.

```
typedef struct DIPROPRANGE {
    DIPROPHEADER diph;
    LONG        lMin;
    LONG        lMax;
} DIPROPRANGE, *LPDIPROPRANGE;

typedef const DIPROPRANGE *LPCDIPROPRANGE;
```

Members

diph

DIPROPHEADER structure that must be initialized as follows:

Member	Value
dwSize	**sizeof(DIPROPRANGE)**
dwHeaderSize	**sizeof(DIPROPHEADER)**
dwObj	If the **dwHow** member is DIPH_DEVICE, this member must be 0. If the **dwHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. If the **dwHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. For example, if the *c_dfDIMouse* data format is selected, it must be one of the DIMOFS_* values.Identifier of the object whose property is being retrieved or set. If the **dwHow** member is DIPH_BYUSAGE, the device must be a Human Interface Device (HID). The device object will be identified by the HID usage page and usage values in packed form.
dwHow	Specifies how the **dwObj** member should be interpreted. See the description of the **dwObj** member in this table for details.

lMin

Lower limit of the range. If the range of the device is unrestricted, this value is DIPROPRANGE_NOMIN when the **IDirectInputDevice8::GetProperty** method returns.

lMax

Upper limit of the range. If the range of the device is unrestricted, this value is DIPROPRANGE_NOMAX when the **IDirectInputDevice8::GetProperty** method returns.

Remarks

The range values for devices whose ranges are unrestricted wraparound.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

See Also

IDirectInputDevice8::GetProperty, **IDirectInputDevice8::SetProperty**

DIPROPSTRING

Gets and retrieves properties as strings.

```
typedef struct DIPROPSTRING {
   DIPROPHEADER diph;
   WCHAR        wsz[MAX_PATH];
} DIPROPSTRING, * LPDIPROPSTRING;;

typedef const DIPROPSTRING *LPCDIPROPSTRING;
```

Members

diph
DIPROPHEADER structure that must be initialized as follows:

Member	Value
dwSize	**sizeof(DIPROPSTRING)**
dwHeaderSize	**sizeof(DIPROPHEADER)**
dwObj	If the **dwHow** member is DIPH_DEVICE, this member must be 0. If the **dwHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. If the **dwHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. For example, if the

	c_dfDIMouse data format is selected, it must be one of the DIMOFS_* values. If the **dwHow** member is DIPH_BYUSAGE, the device must be a Human Interface Device (HID). The device object will be identified by the HID usage page and usage values in packed form.
dwHow	Specifies how the **dwObj** member should be interpreted. See the description of the **dwObj** member in this table for details.

wsz
Unicode string that specifies or retrieves the property.

Remarks

The DIPROP_INSTANCENAME and DIPROP_PRODUCTNAME properties associated with the **DIPROPSTRING** structure enable advanced applications to perform operations on a Human Interface Device (HID) that are not supported by Microsoft® DirectInput®.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 98.
Header: Declared in Dinput.h.

DIRAMPFORCE

Contains type-specific information for effects that are marked as **DIEFT_RAMPFORCE**. The structure describes a ramp force effect.

A pointer to a single **DIRAMPFORCE** structure for an effect is passed in the **lpvTypeSpecificParams** member of the **DIEFFECT** structure.

```
typedef struct DIRAMPFORCE {
    LONG lStart;
    LONG lEnd;
} DIRAMPFORCE, *LPDIRAMPFORCE;

typedef const DIRAMPFORCE *LPCDIRAMPFORCE;
```

Members

lStart
Magnitude at the start of the effect, in the range from –10,000 through 10,000.

lEnd
Magnitude at the end of the effect, in the range from –10,000 through 10,000.

Remarks

The **dwDuration** for a ramp force effect cannot be INFINITE.

Requirements

Windows NT/2000: Requires Windows® 2000.
Windows 95/98: Requires Windows 95 or later. Available as a redistributable for Windows 95.
Header: Declared in Dinput.h.

Device Constants

This section is a reference for constants used to interpret data for keys, buttons, and axes.

- Keyboard Device Constants
- DirectInput and Japanese Keyboards
- Mouse Device Constants
- Joystick Device Constants

Keyboard Device Constants

Keyboard device constants, defined in Dinput.h, represent offsets within a keyboard device's data packet, a 256-byte array. The data at a given offset is associated with a keyboard key. Typically, these values are used in the **dwOfs** member of the **DIDEVICEOBJECTDATA**, **DIOBJECTDATAFORMAT** or **DIDEVICEOBJECTINSTANCE** structures, or as indices when accessing data within the array using array notation.

The standard keyboard device constants are listed in the following table.

Constant	Note
DIK_0	On main keyboard
DIK_1	On main keyboard
DIK_2	On main keyboard
DIK_3	On main keyboard
DIK_4	On main keyboard
DIK_5	On main keyboard
DIK_6	On main keyboard
DIK_7	On main keyboard
DIK_8	On main keyboard

DIK_9	On main keyboard
DIK_A	
DIK_ABNT_C1	On numeric pad of Brazilian keyboards
DIK_ABNT_C2	On numeric pad of Brazilian keyboards
DIK_ADD	PLUS SIGN (+) on numeric keypad
DIK_APOSTROPHE	
DIK_APPS	
DIK_AT	On Japanese keyboard
DIK_AX	On Japanese keyboard
DIK_B	
DIK_BACK	BACKSPACE
DIK_BACKSLASH	
DIK_C	
DIK_CALCULATOR	
DIK_CAPITAL	CAPS LOCK
DIK_COLON	On Japanese keyboard
DIK_COMMA	
DIK_CONVERT	On Japanese keyboard
DIK_D	
DIK_DECIMAL	PERIOD (decimal point) on numeric keypad
DIK_DELETE	
DIK_DIVIDE	Forward slash (/) on numeric keypad
DIK_DOWN	DOWN ARROW
DIK_E	
DIK_END	
DIK_EQUALS	On main keyboard
DIK_ESCAPE	
DIK_F	
DIK_F1	
DIK_F2	
DIK_F3	
DIK_F4	
DIK_F5	
DIK_F6	
DIK_F7	
DIK_F8	
DIK_F9	

DIK_F10	
DIK_F11	
DIK_F12	
DIK_F13	On NEC PC-98 Japanese keyboard
DIK_F14	On NEC PC-98 Japanese keyboard
DIK_F15	On NEC PC-98 Japanese keyboard
DIK_G	
DIK_GRAVE	Grave accent (`)
DIK_H	
DIK_HOME	
DIK_I	
DIK_INSERT	
DIK_J	
DIK_K	
DIK_KANA	On Japanese keyboard
DIK_KANJI	On Japanese keyboard
DIK_L	
DIK_LBRACKET	Left square bracket [
DIK_LCONTROL	Left CTRL
DIK_LEFT	LEFT ARROW
DIK_LMENU	Left ALT
DIK_LSHIFT	Left SHIFT
DIK_LWIN	Left Microsoft® Windows® logo key
DIK_M	
DIK_MAIL	
DIK_MEDIASELECT	Media Select key, which displays a selection of supported media players on the system
DIK_MEDIASTOP	
DIK_MINUS	On main keyboard
DIK_MULTIPLY	Asterisk (*) on numeric keypad
DIK_MUTE	
DIK_MYCOMPUTER	
DIK_N	
DIK_NEXT	PAGE DOWN
DIK_NEXTTRACK	Next track
DIK_NOCONVERT	On Japanese keyboard
DIK_NUMLOCK	

DIK_NUMPAD0	
DIK_NUMPAD1	
DIK_NUMPAD2	
DIK_NUMPAD3	
DIK_NUMPAD4	
DIK_NUMPAD5	
DIK_NUMPAD6	
DIK_NUMPAD7	
DIK_NUMPAD8	
DIK_NUMPAD9	
DIK_NUMPADCOMMA	On numeric keypad of NEC PC-98 Japanese keyboard
DIK_NUMPADENTER	
DIK_NUMPADEQUALS	On numeric keypad of NEC PC-98 Japanese keyboard
DIK_O	
DIK_OEM_102	On British and German keyboards
DIK_P	
DIK_PAUSE	
DIK_PERIOD	On main keyboard
DIK_PLAYPAUSE	
DIK_POWER	
DIK_PREVTRACK	Previous track; circumflex on Japanese keyboard
DIK_PRIOR	PAGE UP
DIK_Q	
DIK_R	
DIK_RBRACKET	Right square bracket]
DIK_RCONTROL	Right CTRL
DIK_RETURN	ENTER on main keyboard
DIK_RIGHT	RIGHT ARROW
DIK_RMENU	Right ALT
DIK_RSHIFT	Right SHIFT
DIK_RWIN	Right Windows logo key
DIK_S	
DIK_SCROLL	SCROLL LOCK
DIK_SEMICOLON	
DIK_SLASH	Forward slash (/) on main keyboard
DIK_SLEEP	

DIK_SPACE	SPACEBAR
DIK_STOP	On NEC PC-98 Japanese keyboard
DIK_SUBTRACT	MINUS SIGN (-) on numeric keypad
DIK_SYSRQ	
DIK_T	
DIK_TAB	
DIK_U	
DIK_UNDERLINE	On NEC PC-98 Japanese keyboard
DIK_UNLABELED	On Japanese keyboard
DIK_UP	UP ARROW
DIK_V	
DIK_VOLUMEDOWN	
DIK_VOLUMEUP	
DIK_W	
DIK_WAKE	
DIK_WEBBACK	
DIK_WEBFAVORITES	Displays the Microsoft Internet Explorer Favorites list, the Windows Favorites folder, or the Netscape Bookmarks list.
DIK_WEBFORWARD	
DIK_WEBHOME	
DIK_WEBREFRESH	
DIK_WEBSEARCH	
DIK_WEBSTOP	
DIK_X	
DIK_Y	
DIK_YEN	On Japanese keyboard
DIK_Z	

The following alternate names are available:

Alternate name	**Regular name**	**Note**
DIK_BACKSPACE	DIK_BACK	BACKSPACE
DIK_CAPSLOCK	DIK_CAPITAL	CAPS LOCK
DIK_CIRCUMFLEX	DIK_PREVTRACK	On Japanese keyboard
DIK_DOWNARROW	DIK_DOWN	On arrow keypad
DIK_LALT	DIK_LMENU	Left ALT
DIK_LEFTARROW	DIK_LEFT	On arrow keypad
DIK_NUMPADMINUS	DIK__SUBTRACT	MINUS SIGN (-) on numeric

		keypad
DIK_NUMPADPERIOD	DIK_DECIMAL	PERIOD (decimal point) on numeric keypad
DIK_NUMPADPLUS	DIK_ADD	PLUS SIGN (+) on numeric keypad
DIK_NUMPADSLASH	DIK__DIVIDE	Forward slash (/) on numeric keypad
DIK_NUMPADSTAR	DIK_MULTIPLY	Asterisk (*) on numeric keypad
DIK_PGDN	DIK_NEXT	On arrow keypad
DIK_PGUP	DIK_PRIOR	On arrow keypad
DIK_RALT	DIK_RMENU	Right ALT
DIK_RIGHTARROW	DIK_RIGHT	On arrow keypad
DIK_UPARROW	DIK_UP	On arrow keypad

For information on Japanese keyboards, see DirectInput and Japanese Keyboards.

DirectInput and Japanese Keyboards

The following chart lists keys that are available on Japanese keyboards but not on standard U.S. keyboards, as well as keys that are available on U.S. keyboards but not on the various Japanese keyboards.

On Microsoft® Windows® 2000 and with some keyboards (such as the NEC PC-98) using other operating systems, the DIK_CAPSLOCK, DIK_KANJI, and DIK_KANA keys are toggle buttons and not push buttons. These generate a down event when first pressed, then generate an up event when pressed a second time.

Japanese Keyboard	Additional keys	Missing keys
DOS/V 106, NEC PC-98 106	DIK_AT, DIK_CIRCUMFLEX, DIK_COLON, DIK_CONVERT, DIK_KANA, DIK_KANJI, DIK_NOCONVERT, DIK_YEN	DIK_APOSTROPHE, DIK_EQUALS, DIK_GRAVE
NEC PC-98	DIK_AT, DIK_CIRCUMFLEX, DIK_COLON, DIK_F13, DIK_F14, DIK_F15, DIK_KANA, DIK_KANJI, DIK_NOCONVERT,	DIK_APOSTROPHE, DIK_BACKSLASH, DIK_EQUALS, DIK_GRAVE, DIK_NUMLOCK, DIK_NUMPADENTER,

	DIK_NUMPADCOMMA, DIK_NUMPADEQUALS, DIK_STOP, DIK_UNDERLINE, DIK_YEN	DIK_RCONTROL, DIK_RMENU, DIK_RSHIFT, DIK_SCROLL
AX	DIK_AX, DIK_CONVERT, DIK_KANJI, DIK_NOCONVERT, DIK_YEN	DIK_RCONTROL, DIK_RMENU
J-3100	DIK_KANA, DIK_KANJI, DIK_NOLABEL, DIK_YEN	DIK_RCONTROL, DIK_RMENU

Mouse Device Constants

Mouse device constants, defined in Dinput.h, represent offsets within a mouse device's data packet, the **DIMOUSESTATE** or **DIMOUSESTATE2** structure. The data at a given offset is associated with a device object (button or axis). Typically, these values are used in the **dwOfs** member of the **DIDEVICEOBJECTDATA**, **DIOBJECTDATAFORMAT** or **DIDEVICEOBJECTINSTANCE** structures.

The mouse device constants are the following:

Constant	Note
DIMOFS_BUTTON0	
DIMOFS_BUTTON1	
DIMOFS_BUTTON2	
DIMOFS_BUTTON3	
DIMOFS_BUTTON4	**DIMOUSESTATE2** only
DIMOFS_BUTTON5	**DIMOUSESTATE2** only
DIMOFS_BUTTON6	**DIMOUSESTATE2** only
DIMOFS_BUTTON7	**DIMOUSESTATE2** only
DIMOFS_X	
DIMOFS_Y	
DIMOFS_Z	

Joystick Device Constants

Joystick device constants represent offsets within a joystick device's data packet, the **DIJOYSTATE** structure. The data at a given offset is associated with a device object; that is, a button or axis. Typically, these values are used in the **dwOfs** member

of the **DIDEVICEOBJECTDATA**, **DIOBJECTDATAFORMAT** or **DIDEVICEOBJECTINSTANCE** structures.

The following macros return a constant indicating the offset of the data for a particular button or axis relative to the beginning of the **DIJOYSTATE** structure.

Constant	**Device object**
DIJOFS_BUTTON0 *to* DIJOFS_BUTTON31 *or* DIJOFS_BUTTON(*n*)	Button
DIJOFS_POV(*n*)	Point-of-view indicator
DIJOFS_RX	X-axis rotation
DIJOFS_RY	Y-axis rotation
DIJOFS_RZ	Z-axis rotation (rudder)
DIJOFS_X	X-axis
DIJOFS_Y	Y-axis
DIJOFS_Z	Z-axis
DIJOFS_SLIDER(*n*)	Slider axis

Action Mapping Constants

This section is a reference to the constants defined in Dinput.h for application genres and controls mapped to actions within each genre.

The control constants, such as DIAXIS_FIGHTINGH_MOVE, are used in the **dwSemantic** member of the **DIACTION** structure to associate the control with an action in the application.

The constants for the genres, such as DIVIRTUAL_FIGHTING_HAND2HAND, are used in the **dwGenre** member of the **DIACTIONFORMAT** structure.

Action mapping constants are provided for application genres in the following categories.

- Action Genres
- Arcade Genres
- CAD Genres
- Control Genres
- Driving Genres
- Flight Genres
- Sports Genres
- Strategy Genres

The **dwSemantic** member of the **DIACTION** structure member can map the action to a particular device object rather than a virtual control. Device object constants are provided in the following categories.

- Keyboard Mapping Constants
- Mouse Mapping Constants
- DirectPlay Voice Mapping Constants
- Any-Control Constants

Action Genres

The following genres contain controls for fighting and action games.

- Hand-to-Hand
- Shooting
- Third-Person Action

Hand-to-Hand

The DIVIRTUAL_FIGHTING_HAND2HAND genre contains controls for a first-person fighting game without guns.

Priority 1 Controls

DIAXIS_FIGHTINGH_LATERAL
DIAXIS_FIGHTINGH_MOVE
DIBUTTON_FIGHTINGH_BLOCK
DIBUTTON_FIGHTINGH_CROUCH
DIBUTTON_FIGHTINGH_JUMP
DIBUTTON_FIGHTINGH_KICK
DIBUTTON_FIGHTINGH_MENU
DIBUTTON_FIGHTINGH_PUNCH
DIBUTTON_FIGHTINGH_SPECIAL1
DIBUTTON_FIGHTINGH_SPECIAL2

Priority 2 Controls

DIAXIS_FIGHTINGH_ROTATE
DIBUTTON_FIGHTINGH_BACKWARD_LINK
DIBUTTON_FIGHTINGH_DEVICE
DIBUTTON_FIGHTINGH_DISPLAY
DIBUTTON_FIGHTINGH_DODGE
DIBUTTON_FIGHTINGH_FORWARD_LINK
DIBUTTON_FIGHTINGH_LEFT_LINK

DIBUTTON_FIGHTINGH_RIGHT_LINK
DIBUTTON_FIGHTINGH_PAUSE
DIBUTTON_FIGHTINGH_SELECT
DIHATSWITCH_FIGHTINGH_SLIDE

Shooting

The DIVIRTUAL_FIGHTING_FPS genre contains controls for a first-person fighting game with guns.

Priority 1 Controls

DIAXIS_FPS_LOOKUPDOWN
DIAXIS_FPS_MOVE
DIAXIS_FPS_ROTATE
DIBUTTON_FPS_APPLY
DIBUTTON_FPS_CROUCH
DIBUTTON_FPS_FIRE
DIBUTTON_FPS_JUMP
DIBUTTON_FPS_MENU
DIBUTTON_FPS_SELECT
DIBUTTON_FPS_STRAFE
DIBUTTON_FPS_WEAPONS

Priority 2 Controls

DIAXIS_FPS_SIDESTEP
DIBUTTON_FPS_BACKWARD_LINK
DIBUTTON_FPS_DEVICE
DIBUTTON_FPS_DISPLAY
DIBUTTON_FPS_DODGE
DIBUTTON_FPS_FIRESECONDARY
DIBUTTON_FPS_FORWARD_LINK
DIBUTTON_FPS_GLANCE_DOWN_LINK
DIBUTTON_FPS_GLANCE_UP_LINK
DIBUTTON_FPS_GLANCEL
DIBUTTON_FPS_GLANCER
DIBUTTON_FPS_PAUSE
DIBUTTON_FPS_ROTATE_LEFT_LINK
DIBUTTON_FPS_ROTATE_RIGHT_LINK
DIHATSWITCH_FPS_GLANCE

Third-Person Action

The DIVIRTUAL_FIGHTING_THIRDPERSON genre contains controls for a third-person action game.

Priority 1 Controls

DIAXIS_TPS_MOVE
DIAXIS_TPS_TURN
DIBUTTON_TPS_ACTION
DIBUTTON_TPS_JUMP
DIBUTTON_TPS_MENU
DIBUTTON_TPS_RUN
DIBUTTON_TPS_SELECT
DIBUTTON_TPS_USE

Priority 2 Controls

DIAXIS_TPS_STEP
DIBUTTON_TPS_BACKWARD_LINK
DIBUTTON_TPS_DEVICE
DIBUTTON_TPS_DODGE
DIBUTTON_TPS_FORWARD_LINK
DIBUTTON_TPS_GLANCE_DOWN_LINK
DIBUTTON_TPS_GLANCE_LEFT_LINK
DIBUTTON_TPS_GLANCE_RIGHT_LINK
DIBUTTON_TPS_GLANCE_UP_LINK
DIBUTTON_TPS_INVENTORY
DIBUTTON_TPS_PAUSE
DIBUTTON_TPS_STEPLEFT
DIBUTTON_TPS_STEPRIGHT
DIBUTTON_TPS_TURN_LEFT_LINK
DIBUTTON_TPS_TURN_RIGHT_LINK
DIBUTTON_TPS_VIEW
DIHATSWITCH_TPS_GLANCE

Arcade Genres

The following genres contain controls for arcade-type games in which the object is to move a figure through a dangerous environment.

- Platform

- Side-to-Side

Platform

The DIVIRTUAL_ARCADE_PLATFORM genre contains controls for an arcade-style game.

Priority 1 Controls

DIAXIS_ARCADEP_LATERAL
DIAXIS_ARCADEP_MOVE
DIBUTTON_ARCADEP_CROUCH
DIBUTTON_ARCADEP_FIRE
DIBUTTON_ARCADEP_JUMP
DIBUTTON_ARCADEP_MENU
DIBUTTON_ARCADEP_SELECT
DIBUTTON_ARCADEP_SPECIAL

Priority 2 Controls

DIBUTTON_ARCADEP_BACK_LINK
DIBUTTON_ARCADEP_DEVICE
DIBUTTON_ARCADEP_FIRESECONDARY
DIBUTTON_ARCADEP_FORWARD_LINK
DIBUTTON_ARCADEP_LEFT_LINK
DIBUTTON_ARCADEP_PAUSE
DIBUTTON_ARCADEP_RIGHT_LINK
DIBUTTON_ARCADEP_VIEW_DOWN_LINK
DIBUTTON_ARCADEP_VIEW_LEFT_LINK
DIBUTTON_ARCADEP_VIEW_RIGHT_LINK
DIBUTTON_ARCADEP_VIEW_UP_LINK
DIHATSWITCH_ARCADEP_VIEW

Side-to-Side

The DIVIRTUAL_ARCADE_SIDE2SIDE genre contains controls for a two-dimensional arcade-style game.

Priority 1 Controls

DIAXIS_ARCADES_LATERAL
DIAXIS_ARCADES_MOVE
DIBUTTON_ARCADES_ATTACK

DIBUTTON_ARCADES_CARRY
DIBUTTON_ARCADES_MENU
DIBUTTON_ARCADES_SELECT
DIBUTTON_ARCADES_SPECIAL
DIBUTTON_ARCADES_THROW

Priority 2 Controls
DIBUTTON_ARCADES_BACK_LINK
DIBUTTON_ARCADES_DEVICE
DIBUTTON_ARCADES_FORWARD_LINK
DIBUTTON_ARCADES_LEFT_LINK
DIBUTTON_ARCADES_PAUSE
DIBUTTON_ARCADES_RIGHT_LINK
DIBUTTON_ARCADES_VIEW_DOWN_LINK
DIBUTTON_ARCADES_VIEW_LEFT_LINK
DIBUTTON_ARCADES_VIEW_RIGHT_LINK
DIBUTTON_ARCADES_VIEW_UP_LINK
DIHATSWITCH_ARCADES_VIEW

CAD Genres

The following genres contain controls for computer-assisted drafting applications.

- 2-D Object
- 3-D Model
- 3-D Navigation
- 3-D Object

2-D Object

The DIVIRTUAL_CAD_2DCONTROL genre contains controls to select and move objects in a two-dimensional drafting environment.

Priority 1 Controls
DIAXIS_2DCONTROL_LATERAL
DIAXIS_2DCONTROL_MOVE
DIAXIS_2DCONTROL_INOUT
DIBUTTON_2DCONTROL_MENU
DIBUTTON_2DCONTROL_SELECT
DIBUTTON_2DCONTROL_SPECIAL
DIBUTTON_2DCONTROL_SPECIAL1

DIBUTTON_2DCONTROL_SPECIAL2

Priority 2 Controls
DIAXIS_2DCONTROL_ROTATEZ
DIBUTTON_2DCONTROL_DEVICE
DIBUTTON_2DCONTROL_DISPLAY
DIBUTTON_2DCONTROL_PAUSE
DIHATSWITCH_2DCONTROL_HATSWITCH

3-D Model

The DIVIRTUAL_CAD_MODEL genre contains controls for modeling three-dimensional objects.

Priority 1 Controls
DIAXIS_CADM_LATERAL
DIAXIS_CADM_MOVE
DIAXIS_CADM_INOUT
DIBUTTON_CADM_MENU
DIBUTTON_CADM_SELECT
DIBUTTON_CADM_SPECIAL
DIBUTTON_CADM_SPECIAL1
DIBUTTON_CADM_SPECIAL2

Priority 2 Controls
DIAXIS_CADM_ROTATEX
DIAXIS_CADM_ROTATEY
DIAXIS_CADM_ROTATEZ
DIBUTTON_CADM_DEVICE
DIBUTTON_CADM_DISPLAY
DIBUTTON_CADM_PAUSE
DIHATSWITCH_CADM_HATSWITCH

3-D Navigation

The DIVIRTUAL_CAD_FLYBY genre contains control for fly-through navigation of three-dimensional environments.

Priority 1 Controls
DIAXIS_CADF_LATERAL

DIAXIS_CADF_MOVE
DIAXIS_CADF_INOUT
DIBUTTON_CADF_MENU
DIBUTTON_CADF_SELECT
DIBUTTON_CADF_SPECIAL
DIBUTTON_CADF_SPECIAL1
DIBUTTON_CADF_SPECIAL2

Priority 2 Controls
DIAXIS_CADF_ROTATEX
DIAXIS_CADF_ROTATEY
DIAXIS_CADF_ROTATEZ
DIBUTTON_CADF_DEVICE
DIBUTTON_CADF_DISPLAY
DIBUTTON_CADF_PAUSE
DIHATSWITCH_CADF_HATSWITCH

3-D Object

The DIVIRTUAL_CAD_3DCONTROL genre contains controls to select and move objects in a three-dimensional drafting environment.

Priority 1 Controls
DIAXIS_3DCONTROL_LATERAL
DIAXIS_3DCONTROL_MOVE
DIAXIS_3DCONTROL_INOUT
DIBUTTON_3DCONTROL_MENU
DIBUTTON_3DCONTROL_SELECT
DIBUTTON_3DCONTROL_SPECIAL
DIBUTTON_3DCONTROL_SPECIAL1
DIBUTTON_3DCONTROL_SPECIAL2

Priority 2 Controls
DIAXIS_3DCONTROL_ROTATEX
DIAXIS_3DCONTROL_ROTATEY
DIAXIS_3DCONTROL_ROTATEZ
DIBUTTON_3DCONTROL_DEVICE
DIBUTTON_3DCONTROL_DISPLAY
DIBUTTON_3DCONTROL_PAUSE
DIHATSWITCH_3DCONTROL_HATSWITCH

Control Genres

The following genres contain controls for use with Web browsers and remote-control devices.

- Browser
- Remote Control

Browser

The DIVIRTUAL_BROWSER_CONTROL genre contains controls for a Web browser.

Priority 1 Controls

DIAXIS_BROWSER_LATERAL
DIAXIS_BROWSER_MOVE
DIAXIS_BROWSER_VIEW
DIBUTTON_BROWSER_MENU
DIBUTTON_BROWSER_REFRESH
DIBUTTON_BROWSER_SELECT

Priority 2 Controls

DIBUTTON_BROWSER_DEVICE
DIBUTTON_BROWSER_FAVORITES
DIBUTTON_BROWSER_HISTORY
DIBUTTON_BROWSER_HOME
DIBUTTON_BROWSER_NEXT
DIBUTTON_BROWSER_PAUSE
DIBUTTON_BROWSER_PREVIOUS
DIBUTTON_BROWSER_PRINT
DIBUTTON_BROWSER_SEARCH
DIBUTTON_BROWSER_STOP

Remote Control

The DIVIRTUAL_REMOTE_CONTROL genre contains controls for remote-control devices used with equipment such as media players.

Priority 1 Controls

DIAXIS_REMOTE_SLIDER
DIBUTTON_REMOTE_CHANGE
DIBUTTON_REMOTE_CUE

DIBUTTON_REMOTE_MENU
DIBUTTON_REMOTE_MUTE
DIBUTTON_REMOTE_PLAY
DIBUTTON_REMOTE_RECORD
DIBUTTON_REMOTE_REVIEW
DIBUTTON_REMOTE_SELECT

Priority 2 Controls
DIAXIS_REMOTE_SLIDER2
DIBUTTON_REMOTE_ADJUST
DIBUTTON_REMOTE_CABLE
DIBUTTON_REMOTE_CD
DIBUTTON_REMOTE_DEVICE
DIBUTTON_REMOTE_DIGIT0
DIBUTTON_REMOTE_DIGIT1
DIBUTTON_REMOTE_DIGIT2
DIBUTTON_REMOTE_DIGIT3
DIBUTTON_REMOTE_DIGIT4
DIBUTTON_REMOTE_DIGIT5
DIBUTTON_REMOTE_DIGIT6
DIBUTTON_REMOTE_DIGIT7
DIBUTTON_REMOTE_DIGIT8
DIBUTTON_REMOTE_DIGIT9
DIBUTTON_REMOTE_DVD
DIBUTTON_REMOTE_PAUSE
DIBUTTON_REMOTE_TUNER
DIBUTTON_REMOTE_TV
DIBUTTON_REMOTE_VCR

Driving Genres

The following genres are used in games where the players control vehicles in racing or combat.

- Combat Racing
- Mechanical Fighting
- Racing
- Tank

Combat Racing

The DIVIRTUAL_DRIVING_COMBAT genre contains controls for a vehicle race game with combat.

Priority 1 Controls

DIAXIS_DRIVINGC_ACCELERATE
DIAXIS_DRIVINGC_BRAKE
DIAXIS_DRIVINGC_STEER
DIBUTTON_DRIVINGC_FIRE
DIBUTTON_DRIVINGC_MENU
DIBUTTON_DRIVINGC_TARGET
DIBUTTON_DRIVINGC_WEAPONS

Priority 2 Controls

DIAXIS_DRIVINGC_ACCEL_AND_BRAKE
DIBUTTON_DRIVINGC_ACCELERATE_LINK
DIBUTTON_DRIVINGC_AIDS
DIBUTTON_DRIVINGC_BRAKE
DIBUTTON_DRIVINGC_DASHBOARD
DIBUTTON_DRIVINGC_DEVICE
DIBUTTON_DRIVINGC_FIRESECONDARY
DIBUTTON_DRIVINGC_GLANCE_LEFT_LINK
DIBUTTON_DRIVINGC_GLANCE_RIGHT_LINK
DIBUTTON_DRIVINGC_PAUSE
DIBUTTON_DRIVINGC_SHIFTDOWN
DIBUTTON_DRIVINGC_SHIFTUP
DIBUTTON_DRIVINGC_STEER_LEFT_LINK
DIBUTTON_DRIVINGC_STEER_RIGHT_LINK
DIHATSWITCH_DRIVINGC_GLANCE

Mechanical Fighting

The DIVIRTUAL_DRIVING_MECHA genre contains controls for a walking vehicle in a mechanized-warfare game.

Priority 1 Controls

DIAXIS_MECHA_ROTATE
DIAXIS_MECHA_STEER
DIAXIS_MECHA_THROTTLE
DIAXIS_MECHA_TORSO

DIBUTTON_MECHA_FIRE
DIBUTTON_MECHA_JUMP
DIBUTTON_MECHA_MENU
DIBUTTON_MECHA_REVERSE
DIBUTTON_MECHA_TARGET
DIBUTTON_MECHA_WEAPONS
DIBUTTON_MECHA_ZOOM

Priority 2 Controls
DIBUTTON_MECHA_BACK_LINK
DIBUTTON_MECHA_CENTER
DIBUTTON_MECHA_DEVICE
DIBUTTON_MECHA_FASTER_LINK
DIBUTTON_MECHA_FIRESECONDARY
DIBUTTON_MECHA_FORWARD_LINK
DIBUTTON_MECHA_LEFT_LINK
DIBUTTON_MECHA_PAUSE
DIBUTTON_MECHA_RIGHT_LINK
DIBUTTON_MECHA_ROTATE_LEFT_LINK
DIBUTTON_MECHA_ROTATE_RIGHT_LINK
DIBUTTON_MECHA_SLOWER_LINK
DIBUTTON_MECHA_VIEW
DIHATSWITCH_MECHA_GLANCE

Racing

The DIVIRTUAL_DRIVING_RACE genre contains controls for an auto-racing game without combat.

Priority 1 Controls
DIAXIS_DRIVINGR_ACCELERATE
DIAXIS_DRIVINGR_BRAKE
DIAXIS_DRIVINGR_STEER
DIBUTTON_DRIVINGR_MENU
DIBUTTON_DRIVINGR_SHIFTDOWN
DIBUTTON_DRIVINGR_SHIFTUP
DIBUTTON_DRIVINGR_VIEW

Priority 2 Controls
DIAXIS_DRIVINGR_ACCEL_AND_BRAKE

DIBUTTON_DRIVINGR_ACCELERATE_LINK
DIBUTTON_DRIVINGR_AIDS
DIBUTTON_DRIVINGR_BOOST
DIBUTTON_DRIVINGR_BRAKE
DIBUTTON_DRIVINGR_DASHBOARD
DIBUTTON_DRIVINGR_DEVICE
DIBUTTON_DRIVINGR_GLANCE_LEFT_LINK
DIBUTTON_DRIVINGR_GLANCE_RIGHT_LINK
DIBUTTON_DRIVINGR_MAP
DIBUTTON_DRIVINGR_PIT
DIBUTTON_DRIVINGR_PAUSE
DIBUTTON_DRIVINGR_STEER_LEFT_LINK
DIBUTTON_DRIVINGR_STEER_RIGHT_LINK
DIHATSWITCH_DRIVINGR_GLANCE

Tank

The DIVIRTUAL_DRIVING_TANK genre contains controls for a military tank simulation.

Priority 1 Controls
DIAXIS_DRIVINGT_ACCELERATE
DIAXIS_DRIVINGT_BARREL
DIAXIS_DRIVINGT_ROTATE
DIAXIS_DRIVINGT_STEER
DIBUTTON_DRIVINGT_FIRE
DIBUTTON_DRIVINGT_MENU
DIBUTTON_DRIVINGT_TARGET
DIBUTTON_DRIVINGT_WEAPONS

Priority 2 Controls
DIAXIS_DRIVINGT_ACCEL_AND_BRAKE
DIAXIS_DRIVINGT_BRAKE
DIBUTTON_DRIVINGT_ACCELERATE_LINK
DIBUTTON_DRIVINGT_BARREL_UP_LINK
DIBUTTON_DRIVINGT_BARREL_DOWN_LINK
DIBUTTON_DRIVINGT_BRAKE
DIBUTTON_DRIVINGT_DASHBOARD
DIBUTTON_DRIVINGT_DEVICE
DIBUTTON_DRIVINGT_FIRESECONDARY

DIBUTTON_DRIVINGT_GLANCE_LEFT_LINK
DIBUTTON_DRIVINGT_GLANCE_RIGHT_LINK
DIBUTTON_DRIVINGT_PAUSE
DIBUTTON_DRIVINGT_ROTATE_LEFT_LINK
DIBUTTON_DRIVINGT_ROTATE_RIGHT_LINK
DIBUTTON_DRIVINGT_STEER_LEFT_LINK
DIBUTTON_DRIVINGT_STEER_RIGHT_LINK
DIBUTTON_DRIVINGT_VIEW
DIHATSWITCH_DRIVINGT_GLANCE

Flight Genres

The following genres are used in games and simulations where the players control aircraft or spaceships.

- Air Combat
- Civilian Flight
- Helicopter Combat
- Space Combat

Air Combat

The DIVIRTUAL_FLYING_MILITARY genre contains controls for a fixed-wing aerial combat simulation.

Priority 1 Controls
DIAXIS_FLYINGM_BANK
DIAXIS_FLYINGM_PITCH
DIAXIS_FLYINGM_THROTTLE
DIBUTTON_FLYINGM_FIRE
DIBUTTON_FLYINGM_MENU
DIBUTTON_FLYINGM_TARGET
DIBUTTON_FLYINGM_WEAPONS

Priority 2 Controls
DIAXIS_FLYINGM_BRAKE
DIAXIS_FLYINGM_FLAPS
DIAXIS_FLYINGM_RUDDER
DIBUTTON_FLYINGM_BRAKE_LINK
DIBUTTON_FLYINGM_COUNTER

DIBUTTON_FLYINGM_DEVICE
DIBUTTON_FLYINGM_DISPLAY
DIBUTTON_FLYINGM_FASTER_LINK
DIBUTTON_FLYINGM_FIRESECONDARY
DIBUTTON_FLYINGM_FLAPSDOWN
DIBUTTON_FLYINGM_FLAPSUP
DIBUTTON_FLYINGM_GEAR
DIBUTTON_FLYINGM_GLANCE_DOWN_LINK
DIBUTTON_FLYINGM_GLANCE_LEFT_LINK
DIBUTTON_FLYINGM_GLANCE_RIGHT_LINK
DIBUTTON_FLYINGM_GLANCE_UP_LINK
DIBUTTON_FLYINGM_PAUSE
DIBUTTON_FLYINGM_SLOWER_LINK
DIBUTTON_FLYINGM_VIEW
DIHATSWITCH_FLYINGM_GLANCE

Civilian Flight

The DIVIRTUAL_FLYING_CIVILIAN genre contains controls for noncombat flight simulations.

Priority 1 Controls

DIAXIS_FLYINGC_BANK
DIAXIS_FLYINGC_PITCH
DIAXIS_FLYINGC_THROTTLE
DIBUTTON_FLYINGC_DISPLAY
DIBUTTON_FLYINGC_GEAR
DIBUTTON_FLYINGC_MENU
DIBUTTON_FLYINGC_VIEW

Priority 2 Controls

DIAXIS_FLYINGC_BRAKE
DIAXIS_FLYINGC_FLAPS
DIAXIS_FLYINGC_RUDDER
DIBUTTON_FLYINGC_BRAKE_LINK
DIBUTTON_FLYINGC_DEVICE
DIBUTTON_FLYINGC_FASTER_LINK
DIBUTTON_FLYINGC_FLAPSDOWN
DIBUTTON_FLYINGC_FLAPSUP
DIBUTTON_FLYINGC_GLANCE_DOWN_LINK

DIBUTTON_FLYINGC_GLANCE_LEFT_LINK
DIBUTTON_FLYINGC_GLANCE_RIGHT_LINK
DIBUTTON_FLYINGC_GLANCE_UP_LINK
DIBUTTON_FLYINGC_PAUSE
DIBUTTON_FLYINGC_SLOWER_LINK
DIHATSWITCH_FLYINGC_GLANCE

Helicopter Combat

The DIVIRTUAL_FLYING_HELICOPTER genre contains controls for a combat helicopter simulation.

Priority 1 Controls

DIAXIS_FLYINGH_BANK
DIAXIS_FLYINGH_COLLECTIVE
DIAXIS_FLYINGH_PITCH
DIBUTTON_FLYINGH_FIRE
DIBUTTON_FLYINGH_MENU
DIBUTTON_FLYINGH_TARGET
DIBUTTON_FLYINGH_WEAPONS

Priority 2 Controls

DIAXIS_FLYINGH_THROTTLE
DIAXIS_FLYINGH_TORQUE
DIBUTTON_FLYINGH_COUNTER
DIBUTTON_FLYINGH_DEVICE
DIBUTTON_FLYINGH_FASTER_LINK
DIBUTTON_FLYINGH_FIRESECONDARY
DIBUTTON_FLYINGH_GEAR
DIBUTTON_FLYINGH_GLANCE_DOWN_LINK
DIBUTTON_FLYINGH_GLANCE_LEFT_LINK
DIBUTTON_FLYINGH_GLANCE_RIGHT_LINK
DIBUTTON_FLYINGH_GLANCE_UP_LINK
DIBUTTON_FLYINGH_PAUSE
DIBUTTON_FLYINGH_SLOWER_LINK
DIBUTTON_FLYINGH_VIEW
DIHATSWITCH_FLYINGH_GLANCE

Space Combat

The DIVIRTUAL_SPACESIM genre contains controls for a spaceship combat simulation.

Priority 1 Controls

DIAXIS_SPACESIM_LATERAL
DIAXIS_SPACESIM_MOVE
DIAXIS_SPACESIM_THROTTLE
DIBUTTON_SPACESIM_FIRE
DIBUTTON_SPACESIM_MENU
DIBUTTON_SPACESIM_TARGET
DIBUTTON_SPACESIM_WEAPONS

Priority 2 Controls

DIAXIS_SPACESIM_CLIMB
DIAXIS_SPACESIM_ROTATE
DIBUTTON_SPACESIM_BACKWARD_LINK
DIBUTTON_SPACESIM_DEVICE
DIBUTTON_SPACESIM_DISPLAY
DIBUTTON_SPACESIM_FASTER_LINK
DIBUTTON_SPACESIM_FIRESECONDARY
DIBUTTON_SPACESIM_FORWARD_LINK
DIBUTTON_SPACESIM_GEAR
DIBUTTON_SPACESIM_GLANCE_DOWN_LINK
DIBUTTON_SPACESIM_GLANCE_LEFT_LINK
DIBUTTON_SPACESIM_GLANCE_RIGHT_LINK
DIBUTTON_SPACESIM_GLANCE_UP_LINK
DIBUTTON_SPACESIM_LEFT_LINK
DIBUTTON_SPACESIM_LOWER
DIBUTTON_SPACESIM_PAUSE
DIBUTTON_SPACESIM_RAISE
DIBUTTON_SPACESIM_RIGHT_LINK
DIBUTTON_SPACESIM_SLOWER_LINK
DIBUTTON_SPACESIM_TURN_LEFT_LINK
DIBUTTON_SPACESIM_TURN_RIGHT_LINK
DIBUTTON_SPACESIM_VIEW
DIHATSWITCH_SPACESIM_GLANCE

Sports Genres

The following genres contain controls for sports games and simulations.

- Baseball Batting
- Baseball Fielding
- Baseball Pitching
- Basketball Defense
- Basketball Offense
- Fishing
- Football Defense
- Football Offense
- Football Play
- Football Quarterback
- Golf
- Hockey Defense
- Hockey Goalie
- Hockey Offense
- Hunting
- Mountain Biking
- Racquet
- Skiing
- Soccer Defense
- Soccer Offense

Baseball Batting

The DIVIRTUAL_SPORTS_BASEBALL_BAT genre contains controls for a batter in a baseball game.

Priority 1 Controls

DIAXIS_BASEBALLB_LATERAL
DIAXIS_BASEBALLB_MOVE
DIBUTTON_BASEBALLB_BUNT
DIBUTTON_BASEBALLB_BURST
DIBUTTON_BASEBALLB_CONTACT
DIBUTTON_BASEBALLB_MENU
DIBUTTON_BASEBALLB_NORMAL
DIBUTTON_BASEBALLB_POWER

DIBUTTON_BASEBALLB_SELECT
DIBUTTON_BASEBALLB_SLIDE
DIBUTTON_BASEBALLB_STEAL

Priority 2 Controls
DIBUTTON_BASEBALLB_BOX
DIBUTTON_BASEBALLB_BACK_LINK
DIBUTTON_BASEBALLB_DEVICE
DIBUTTON_BASEBALLB_FORWARD_LINK
DIBUTTON_BASEBALLB_LEFT_LINK
DIBUTTON_BASEBALLB_NOSTEAL
DIBUTTON_BASEBALLB_PAUSE
DIBUTTON_BASEBALLB_RIGHT_LINK

Baseball Fielding

The DIVIRTUAL_SPORTS_BASEBALL_FIELD genre contains controls for fielders in a baseball game.

Priority 1 Controls
DIAXIS_BASEBALLF_LATERAL
DIAXIS_BASEBALLF_MOVE
DIBUTTON_BASEBALLF_BURST
DIBUTTON_BASEBALLF_DIVE
DIBUTTON_BASEBALLF_JUMP
DIBUTTON_BASEBALLF_MENU
DIBUTTON_BASEBALLF_NEAREST
DIBUTTON_BASEBALLF_THROW1
DIBUTTON_BASEBALLF_THROW2

Priority 2 Controls
DIBUTTON_BASEBALLF_AIM_LEFT_LINK
DIBUTTON_BASEBALLF_AIM_RIGHT_LINK
DIBUTTON_BASEBALLF_BACK_LINK
DIBUTTON_BASEBALLF_DEVICE
DIBUTTON_BASEBALLF_FORWARD_LINK
DIBUTTON_BASEBALLF_PAUSE
DIBUTTON_BASEBALLF_SHIFTIN
DIBUTTON_BASEBALLF_SHIFTOUT

Baseball Pitching

The DIVIRTUAL_SPORTS_BASEBALL_PITCH genre contains controls for a pitcher in a baseball game.

Priority 1 Controls

DIAXIS_BASEBALLP_LATERAL
DIAXIS_BASEBALLP_MOVE
DIBUTTON_BASEBALLP_BASE
DIBUTTON_BASEBALLP_FAKE
DIBUTTON_BASEBALLP_MENU
DIBUTTON_BASEBALLP_PITCH
DIBUTTON_BASEBALLP_SELECT
DIBUTTON_BASEBALLP_THROW

Priority 2 Controls

DIBUTTON_BASEBALLP_BACK_LINK
DIBUTTON_BASEBALLP_DEVICE
DIBUTTON_BASEBALLP_FORWARD_LINK
DIBUTTON_BASEBALLP_LEFT_LINK
DIBUTTON_BASEBALLP_LOOK
DIBUTTON_BASEBALLP_PAUSE
DIBUTTON_BASEBALLP_RIGHT_LINK
DIBUTTON_BASEBALLP_WALK

Basketball Defense

The DIVIRTUAL_SPORTS_BASKETBALL_DEFENSE genre contains controls for defensive moves in a basketball game.

Priority 1 Controls

DIAXIS_BBALLD_LATERAL
DIAXIS_BBALLD_MOVE
DIBUTTON_BBALLD_BURST
DIBUTTON_BBALLD_FAKE
DIBUTTON_BBALLD_JUMP
DIBUTTON_BBALLD_MENU
DIBUTTON_BBALLD_PLAY
DIBUTTON_BBALLD_PLAYER
DIBUTTON_BBALLD_SPECIAL
DIBUTTON_BBALLD_STEAL

Priority 2 Controls
DIBUTTON_BBALLD_BACK_LINK
DIBUTTON_BBALLD_DEVICE
DIBUTTON_BBALLD_FORWARD_LINK
DIBUTTON_BBALLD_LEFT_LINK
DIBUTTON_BBALLD_PAUSE
DIBUTTON_BBALLD_RIGHT_LINK
DIBUTTON_BBALLD_SUBSTITUTE
DIBUTTON_BBALLD_TIMEOUT
DIHATSWITCH_BBALLD_GLANCE

Basketball Offense

The DIVIRTUAL_SPORTS_BASKETBALL_OFFENSE genre contains controls for the player who has the ball in a basketball game.

Priority 1 Controls
DIAXIS_BBALLO_LATERAL
DIAXIS_BBALLO_MOVE
DIBUTTON_BBALLO_BURST
DIBUTTON_BBALLO_CALL
DIBUTTON_BBALLO_DUNK
DIBUTTON_BBALLO_FAKE
DIBUTTON_BBALLO_MENU
DIBUTTON_BBALLO_PASS
DIBUTTON_BBALLO_PLAYER
DIBUTTON_BBALLO_SHOOT
DIBUTTON_BBALLO_SPECIAL

Priority 2 Controls
DIBUTTON_BBALLO_BACK_LINK
DIBUTTON_BBALLO_DEVICE
DIBUTTON_BBALLO_FORWARD_LINK
DIBUTTON_BBALLO_JAB
DIBUTTON_BBALLO_LEFT_LINK
DIBUTTON_BBALLO_PAUSE
DIBUTTON_BBALLO_PLAY
DIBUTTON_BBALLO_POST
DIBUTTON_BBALLO_RIGHT_LINK

DIBUTTON_BBALLO_SCREEN
DIBUTTON_BBALLO_SUBSTITUTE
DIBUTTON_BBALLO_TIMEOUT
DIHATSWITCH_BBALLO_GLANCE

Fishing

The DIVIRTUAL_SPORTS_FISHING genre contains controls for a sport-fishing simulation.

Priority 1 Controls
DIAXIS_FISHING_LATERAL
DIAXIS_FISHING_MOVE
DIBUTTON_FISHING_BAIT
DIBUTTON_FISHING_BINOCULAR
DIBUTTON_FISHING_CAST
DIBUTTON_FISHING_MAP
DIBUTTON_FISHING_MENU
DIBUTTON_FISHING_TYPE

Priority 2 Controls
DIAXIS_FISHING_ROTATE
DIBUTTON_FISHING_BACK_LINK
DIBUTTON_FISHING_CROUCH
DIBUTTON_FISHING_DEVICE
DIBUTTON_FISHING_DISPLAY
DIBUTTON_FISHING_FORWARD_LINK
DIBUTTON_FISHING_JUMP
DIBUTTON_FISHING_LEFT_LINK
DIBUTTON_FISHING_PAUSE
DIBUTTON_FISHING_RIGHT_LINK
DIBUTTON_FISHING_ROTATE_LEFT_LINK
DIBUTTON_FISHING_ROTATE_RIGHT_LINK
DIHATSWITCH_FISHING_GLANCE

Football Defense

The DIVIRTUAL_SPORTS_FOOTBALL_DEFENSE genre contains controls for the defensive side in a North American football game.

Priority 1 Controls

DIAXIS_FOOTBALLD_LATERAL
DIAXIS_FOOTBALLD_MOVE
DIBUTTON_FOOTBALLD_FAKE
DIBUTTON_FOOTBALLD_JUMP
DIBUTTON_FOOTBALLD_MENU
DIBUTTON_FOOTBALLD_PLAY
DIBUTTON_FOOTBALLD_SELECT
DIBUTTON_FOOTBALLD_SUPERTACKLE
DIBUTTON_FOOTBALLD_TACKLE

Priority 2 Controls
DIBUTTON_FOOTBALLD_AUDIBLE
DIBUTTON_FOOTBALLD_BACK_LINK
DIBUTTON_FOOTBALLD_BULLRUSH
DIBUTTON_FOOTBALLD_DEVICE
DIBUTTON_FOOTBALLD_FORWARD_LINK
DIBUTTON_FOOTBALLD_LEFT_LINK
DIBUTTON_FOOTBALLD_PAUSE
DIBUTTON_FOOTBALLD_RIGHT_LINK
DIBUTTON_FOOTBALLD_RIP
DIBUTTON_FOOTBALLD_SPIN
DIBUTTON_FOOTBALLD_SUBSTITUTE
DIBUTTON_FOOTBALLD_SWIM
DIBUTTON_FOOTBALLD_ZOOM

Football Offense

The DIVIRTUAL_SPORTS_FOOTBALL_OFFENSE genre contains controls for the ball carrier in a North American football game.

Priority 1 Controls
DIAXIS_FOOTBALLO_LATERAL
DIAXIS_FOOTBALLO_MOVE
DIBUTTON_FOOTBALLO_JUMP
DIBUTTON_FOOTBALLO_LEFTARM
DIBUTTON_FOOTBALLO_MENU
DIBUTTON_FOOTBALLO_RIGHTARM
DIBUTTON_FOOTBALLO_SPIN
DIBUTTON_FOOTBALLO_THROW

Priority 2 Controls

DIBUTTON_FOOTBALLO_BACK_LINK
DIBUTTON_FOOTBALLO_DEVICE
DIBUTTON_FOOTBALLO_DIVE
DIBUTTON_FOOTBALLO_FORWARD_LINK
DIBUTTON_FOOTBALLO_JUKE
DIBUTTON_FOOTBALLO_LEFT_LINK
DIBUTTON_FOOTBALLO_PAUSE
DIBUTTON_FOOTBALLO_RIGHT_LINK
DIBUTTON_FOOTBALLO_SHOULDER
DIBUTTON_FOOTBALLO_SUBSTITUTE
DIBUTTON_FOOTBALLO_TURBO
DIBUTTON_FOOTBALLO_ZOOM

Football Play

The DIVIRTUAL_SPORTS_FOOTBALL_FIELD genre contains general controls for a North American football game.

Priority 1 Controls

DIBUTTON_FOOTBALLP_HELP
DIBUTTON_FOOTBALLP_MENU
DIBUTTON_FOOTBALLP_PLAY
DIBUTTON_FOOTBALLP_SELECT

Priority 2 Controls

DIBUTTON_FOOTBALLP_DEVICE
DIBUTTON_FOOTBALLP_PAUSE

Football Quarterback

The DIVIRTUAL_SPORTS_FOOTBALL_QBCK genre contains controls for the quarterback in a North American football game.

Priority 1 Controls

DIAXIS_FOOTBALLQ_LATERAL
DIAXIS_FOOTBALLQ_MOVE
DIBUTTON_FOOTBALLQ_FAKE
DIBUTTON_FOOTBALLQ_JUMP
DIBUTTON_FOOTBALLQ_MENU
DIBUTTON_FOOTBALLQ_PASS

DIBUTTON_FOOTBALLQ_SELECT
DIBUTTON_FOOTBALLQ_SLIDE
DIBUTTON_FOOTBALLQ_SNAP

Priority 2 Controls
DIBUTTON_FOOTBALLQ_AUDIBLE
DIBUTTON_FOOTBALLQ_BACK_LINK
DIBUTTON_FOOTBALLQ_DEVICE
DIBUTTON_FOOTBALLQ_FAKESNAP
DIBUTTON_FOOTBALLQ_FORWARD_LINK
DIBUTTON_FOOTBALLQ_LEFT_LINK
DIBUTTON_FOOTBALLQ_MOTION
DIBUTTON_FOOTBALLQ_PAUSE
DIBUTTON_FOOTBALLQ_RIGHT_LINK

Golf

The DIVIRTUAL_SPORTS_GOLF genre contains controls for a golf game.

Priority 1 Controls
DIAXIS_GOLF_LATERAL
DIAXIS_GOLF_MOVE
DIBUTTON_GOLF_DOWN
DIBUTTON_GOLF_FLYBY
DIBUTTON_GOLF_MENU
DIBUTTON_GOLF_SELECT
DIBUTTON_GOLF_SWING
DIBUTTON_GOLF_TERRAIN
DIBUTTON_GOLF_UP

Priority 2 Controls
DIBUTTON_GOLF_BACK_LINK
DIBUTTON_GOLF_DEVICE
DIBUTTON_GOLF_FORWARD_LINK
DIBUTTON_GOLF_LEFT_LINK
DIBUTTON_GOLF_PAUSE
DIBUTTON_GOLF_RIGHT_LINK
DIBUTTON_GOLF_SUBSTITUTE
DIBUTTON_GOLF_TIMEOUT
DIBUTTON_GOLF_ZOOM

DIHATSWITCH_GOLF_SCROLL

Hockey Defense

The DIVIRTUAL_SPORTS_HOCKEY_DEFENSE genre contains controls for defensive moves in a hockey game.

Priority 1 Controls

DIAXIS_HOCKEYD_LATERAL
DIAXIS_HOCKEYD_MOVE
DIBUTTON_HOCKEYD_BLOCK
DIBUTTON_HOCKEYD_BURST
DIBUTTON_HOCKEYD_FAKE
DIBUTTON_HOCKEYD_MENU
DIBUTTON_HOCKEYD_PLAYER
DIBUTTON_HOCKEYD_STEAL

Priority 2 Controls

DIBUTTON_HOCKEYD_BACK_LINK
DIBUTTON_HOCKEYD_DEVICE
DIBUTTON_HOCKEYO_FORWARD_LINK
DIBUTTON_HOCKEYO_LEFT_LINK
DIBUTTON_HOCKEYD_PAUSE
DIBUTTON_HOCKEYO_RIGHT_LINK
DIBUTTON_HOCKEYD_STRATEGY
DIBUTTON_HOCKEYD_SUBSTITUTE
DIBUTTON_HOCKEYD_TIMEOUT
DIBUTTON_HOCKEYD_ZOOM
DIHATSWITCH_HOCKEYD_SCROLL

Hockey Goalie

The DIVIRTUAL_SPORTS_HOCKEY_GOALIE genre contains controls for a goalkeeper in a hockey game.

Priority 1 Controls

DIAXIS_HOCKEYG_LATERAL
DIAXIS_HOCKEYG_MOVE
DIBUTTON_HOCKEYG_BLOCK
DIBUTTON_HOCKEYG_MENU
DIBUTTON_HOCKEYG_PASS

DIBUTTON_HOCKEYG_POKE
DIBUTTON_HOCKEYG_STEAL

Priority 2 Controls
DIBUTTON_HOCKEYG_BACK_LINK
DIBUTTON_HOCKEYG_DEVICE
DIBUTTON_HOCKEYG_FORWARD_LINK
DIBUTTON_HOCKEYG_LEFT_LINK
DIBUTTON_HOCKEYG_PAUSE
DIBUTTON_HOCKEYG_RIGHT_LINK
DIBUTTON_HOCKEYG_STRATEGY
DIBUTTON_HOCKEYG_SUBSTITUTE
DIBUTTON_HOCKEYG_TIMEOUT
DIBUTTON_HOCKEYG_ZOOM
DIHATSWITCH_HOCKEYG_SCROLL

Hockey Offense

The DIVIRTUAL_SPORTS_HOCKEY_OFFENSE genre contains controls for offensive moves in a hockey game.

Priority 1 Controls
DIAXIS_HOCKEYO_LATERAL
DIAXIS_HOCKEYO_MOVE
DIBUTTON_HOCKEYO_BURST
DIBUTTON_HOCKEYO_FAKE
DIBUTTON_HOCKEYO_MENU
DIBUTTON_HOCKEYO_PASS
DIBUTTON_HOCKEYO_SHOOT
DIBUTTON_HOCKEYO_SPECIAL

Priority 2 Controls
DIBUTTON_HOCKEYO_BACK_LINK
DIBUTTON_HOCKEYO_DEVICE
DIBUTTON_HOCKEYO_FORWARD_LINK
DIBUTTON_HOCKEYO_LEFT_LINK
DIBUTTON_HOCKEYO_PAUSE
DIBUTTON_HOCKEYO_RIGHT_LINK
DIBUTTON_HOCKEYO_STRATEGY
DIBUTTON_HOCKEYO_SUBSTITUTE

DIBUTTON_HOCKEYO_TIMEOUT
DIBUTTON_HOCKEYO_ZOOM
DIHATSWITCH_HOCKEYO_SCROLL

Hunting

The DIVIRTUAL_SPORTS_HUNTING genre contains controls for a hunting simulation.

Priority 1 Controls

DIAXIS_HUNTING_LATERAL
DIAXIS_HUNTING_MOVE
DIBUTTON_HUNTING_AIM
DIBUTTON_HUNTING_BINOCULAR
DIBUTTON_HUNTING_CALL
DIBUTTON_HUNTING_FIRE
DIBUTTON_HUNTING_MAP
DIBUTTON_HUNTING_MENU
DIBUTTON_HUNTING_SPECIAL
DIBUTTON_HUNTING_WEAPON

Priority 2 Controls

DIAXIS_HUNTING_ROTATE
DIBUTTON_HUNTING_BACK_LINK
DIBUTTON_HUNTING_CROUCH
DIBUTTON_HUNTING_DEVICE
DIBUTTON_HUNTING_DISPLAY
DIBUTTON_HUNTING_FIRESECONDARY
DIBUTTON_HUNTING_FORWARD_LINK
DIBUTTON_HUNTING_JUMP
DIBUTTON_HUNTING_LEFT_LINK
DIBUTTON_HUNTING_PAUSE
DIBUTTON_HUNTING_RIGHT_LINK
DIBUTTON_HUNTING_ROTATE_LEFT_LINK
DIBUTTON_HUNTING_ROTATE_RIGHT_LINK
DIHATSWITCH_HUNTING_GLANCE

Mountain Biking

The DIVIRTUAL_SPORTS_BIKING_MOUNTAIN contains controls for a mountain-bike game.

Priority 1 Controls

DIAXIS_BIKINGM_PEDAL
DIAXIS_BIKINGM_TURN
DIBUTTON_BIKINGM_CAMERA
DIBUTTON_BIKINGM_JUMP
DIBUTTON_BIKINGM_MENU
DIBUTTON_BIKINGM_SELECT
DIBUTTON_BIKINGM_SPECIAL1
DIBUTTON_BIKINGM_SPECIAL2

Priority 2 Controls

DIAXIS_BIKINGM_BRAKE
DIBUTTON_BIKINGM_BRAKE_BUTTON_LINK
DIBUTTON_BIKINGM_DEVICE
DIBUTTON_BIKINGM_FASTER_LINK
DIBUTTON_BIKINGM_LEFT_LINK
DIBUTTON_BIKINGM_PAUSE
DIBUTTON_BIKINGM_RIGHT_LINK
DIBUTTON_BIKINGM_SLOWER_LINK
DIBUTTON_BIKINGM_ZOOM
DIHATSWITCH_BIKINGM_SCROLL

Racquet

The DIVIRTUAL_SPORTS_RACQUET genre includes racquet games such as tennis, table tennis, and squash.

Priority 1 Controls

DIAXIS_RACQUET_LATERAL
DIAXIS_RACQUET_MOVE
DIBUTTON_RACQUET_BACKSWING
DIBUTTON_RACQUET_MENU
DIBUTTON_RACQUET_SELECT
DIBUTTON_RACQUET_SMASH
DIBUTTON_RACQUET_SPECIAL
DIBUTTON_RACQUET_SWING

Priority 2 Controls
DIBUTTON_RACQUET_BACK_LINK
DIBUTTON_RACQUET_DEVICE
DIBUTTON_RACQUET_FORWARD_LINK
DIBUTTON_RACQUET_LEFT_LINK
DIBUTTON_RACQUET_PAUSE
DIBUTTON_RACQUET_RIGHT_LINK
DIBUTTON_RACQUET_SUBSTITUTE
DIBUTTON_RACQUET_TIMEOUT
DIHATSWITCH_RACQUET_GLANCE

Skiing

The DIVIRTUAL_SPORTS_SKIING contains controls for skiing games as well as for similar sports simulations such as snowboarding and skateboarding.

Priority 1 Controls
DIAXIS_SKIING_SPEED
DIAXIS_SKIING_TURN
DIBUTTON_SKIING_CAMERA
DIBUTTON_SKIING_CROUCH
DIBUTTON_SKIING_JUMP
DIBUTTON_SKIING_MENU
DIBUTTON_SKIING_SELECT
DIBUTTON_SKIING_SPECIAL1
DIBUTTON_SKIING_SPECIAL2

Priority 2 Controls
DIBUTTON_SKIING_DEVICE
DIBUTTON_SKIING_FASTER_LINK
DIBUTTON_SKIING_LEFT_LINK
DIBUTTON_SKIING_PAUSE
DIBUTTON_SKIING_RIGHT_LINK
DIBUTTON_SKIING_SLOWER_LINK
DIBUTTON_SKIING_ZOOM
DIHATSWITCH_SKIING_GLANCE

Soccer Defense

The DIVIRTUAL_SPORTS_SOCCER_DEFENSE contains controls for defensive moves in a soccer (international football) game.

Priority 1 Controls

DIAXIS_SOCCERD_LATERAL
DIAXIS_SOCCERD_MOVE
DIBUTTON_SOCCERD_BLOCK
DIBUTTON_SOCCERD_FAKE
DIBUTTON_SOCCERD_MENU
DIBUTTON_SOCCERD_PLAYER
DIBUTTON_SOCCERD_SELECT
DIBUTTON_SOCCERD_SLIDE
DIBUTTON_SOCCERD_SPECIAL
DIBUTTON_SOCCERD_STEAL

Priority 2 Controls

DIBUTTON_SOCCERD_BACK_LINK
DIBUTTON_SOCCERD_CLEAR
DIBUTTON_SOCCERD_DEVICE
DIBUTTON_SOCCERD_FORWARD_LINK
DIBUTTON_SOCCERD_FOUL
DIBUTTON_SOCCERD_GOALIECHARGE
DIBUTTON_SOCCERD_HEAD
DIBUTTON_SOCCERD_LEFT_LINK
DIBUTTON_SOCCERD_PAUSE
DIBUTTON_SOCCERD_RIGHT_LINK
DIBUTTON_SOCCERD_SUBSTITUTE
DIHATSWITCH_SOCCERD_GLANCE

Soccer Offense

The DIVIRTUAL_SPORTS_SOCCER_OFFENSE contains controls for offensive moves in a soccer (international football) game.

Priority 1 Controls

DIAXIS_SOCCERO_BEND
DIAXIS_SOCCERO_LATERAL
DIAXIS_SOCCERO_MOVE
DIBUTTON_SOCCERO_FAKE

DIBUTTON_SOCCERO_MENU
DIBUTTON_SOCCERO_PASS
DIBUTTON_SOCCERO_PLAYER
DIBUTTON_SOCCERO_SELECT
DIBUTTON_SOCCERO_SHOOT
DIBUTTON_SOCCERO_SPECIAL1

Priority 2 Controls
DIBUTTON_SOCCERO_BACK_LINK
DIBUTTON_SOCCERO_CONTROL
DIBUTTON_SOCCERO_DEVICE
DIBUTTON_SOCCERO_FORWARD_LINK
DIBUTTON_SOCCERO_HEAD
DIBUTTON_SOCCERO_LEFT_LINK
DIBUTTON_SOCCERO_PASSTHRU
DIBUTTON_SOCCERO_PAUSE
DIBUTTON_SOCCERO_RIGHT_LINK
DIBUTTON_SOCCERO_SHOOTHIGH
DIBUTTON_SOCCERO_SHOOTLOW
DIBUTTON_SOCCERO_SPRINT
DIBUTTON_SOCCERO_SUBSTITUTE
DIHATSWITCH_SOCCERO_GLANCE

Strategy Genres

The following genres contain controls for strategy, role-playing, and adventure games.

- Role-Playing
- Turn-Based

Role-Playing

The DIVIRTUAL_STRATEGY_ROLEPLAYING genre contains controls for a role-playing or adventure game in which navigation, problem-solving, and fighting with weapons and magic are common activities.

Priority 1 Controls
DIAXIS_STRATEGYR_LATERAL
DIAXIS_STRATEGYR_MOVE
DIBUTTON_STRATEGYR_APPLY

DIBUTTON_STRATEGYR_ATTACK
DIBUTTON_STRATEGYR_CAST
DIBUTTON_STRATEGYR_CROUCH
DIBUTTON_STRATEGYR_GET
DIBUTTON_STRATEGYR_JUMP
DIBUTTON_STRATEGYR_MENU
DIBUTTON_STRATEGYR_SELECT

Priority 2 Controls
DIAXIS_STRATEGYR_ROTATE
DIBUTTON_STRATEGYR_BACK_LINK
DIBUTTON_STRATEGYR_DEVICE
DIBUTTON_STRATEGYR_DISPLAY
DIBUTTON_STRATEGYR_FORWARD_LINK
DIBUTTON_STRATEGYR_LEFT_LINK
DIBUTTON_STRATEGYR_MAP
DIBUTTON_STRATEGYR_PAUSE
DIBUTTON_STRATEGYR_RIGHT_LINK
DIBUTTON_STRATEGYR_ROTATE_LEFT_LINK
DIBUTTON_STRATEGYR_ROTATE_RIGHT_LINK
DIHATSWITCH_STRATEGYR_GLANCE

Turn-Based

The DIVIRTUAL_STRATEGY_TURN genre contains controls for a turn-based strategy game.

Priority 1 Controls
DIAXIS_STRATEGYT_LATERAL
DIAXIS_STRATEGYT_MOVE
DIBUTTON_STRATEGYT_APPLY
DIBUTTON_STRATEGYT_INSTRUCT
DIBUTTON_STRATEGYT_MENU
DIBUTTON_STRATEGYT_SELECT
DIBUTTON_STRATEGYT_TEAM
DIBUTTON_STRATEGYT_TURN

Priority 2 Controls
DIBUTTON_STRATEGYT_BACK_LINK
DIBUTTON_STRATEGYT_DEVICE

DIBUTTON_STRATEGYT_DISPLAY
DIBUTTON_STRATEGYT_FORWARD_LINK
DIBUTTON_STRATEGYT_LEFT_LINK
DIBUTTON_STRATEGYT_MAP
DIBUTTON_STRATEGYT_PAUSE
DIBUTTON_STRATEGYT_RIGHT_LINK
DIBUTTON_STRATEGYT_ZOOM

Keyboard Mapping Constants

The following constants are used in the **dwSemantic** member of the **DIACTION** structure to map an action to a physical key.

Constant	Note
DIKEYBOARD_0	On main keyboard
DIKEYBOARD_1	On main keyboard
DIKEYBOARD_2	On main keyboard
DIKEYBOARD_3	On main keyboard
DIKEYBOARD_4	On main keyboard
DIKEYBOARD_5	On main keyboard
DIKEYBOARD_6	On main keyboard
DIKEYBOARD_7	On main keyboard
DIKEYBOARD_8	On main keyboard
DIKEYBOARD_9	On main keyboard
DIKEYBOARD_A	
DIKEYBOARD_ABNT_C1	On numeric pad of Brazilian keyboards
DIKEYBOARD_ABNT_C2	On numeric pad of Brazilian keyboards
DIKEYBOARD_ADD	PLUS SIGN (+) on numeric keypad
DIKEYBOARD_APOSTROPHE	
DIKEYBOARD_APPS	
DIKEYBOARD_AT	On Japanese keyboard
DIKEYBOARD_AX	On Japanese keyboard
DIKEYBOARD_B	
DIKEYBOARD_BACK	BACKSPACE
DIKEYBOARD_BACKSLASH	
DIKEYBOARD_C	
DIKEYBOARD_CALCULATOR	
DIKEYBOARD_CAPITAL	CAPS LOCK

DIKEYBOARD_COLON	On Japanese keyboard
DIKEYBOARD_COMMA	
DIKEYBOARD_CONVERT	On Japanese keyboard
DIKEYBOARD_D	
DIKEYBOARD_DECIMAL	PERIOD (decimal point) on numeric keypad
DIKEYBOARD_DELETE	
DIKEYBOARD_DIVIDE	Forward slash (/) on numeric keypad
DIKEYBOARD_DOWN	DOWN ARROW
DIKEYBOARD_E	
DIKEYBOARD_END	
DIKEYBOARD_EQUALS	On main keyboard
DIKEYBOARD_ESCAPE	
DIKEYBOARD_F	
DIKEYBOARD_F1	
DIKEYBOARD_F2	
DIKEYBOARD_F3	
DIKEYBOARD_F4	
DIKEYBOARD_F5	
DIKEYBOARD_F6	
DIKEYBOARD_F7	
DIKEYBOARD_F8	
DIKEYBOARD_F9	
DIKEYBOARD_F10	
DIKEYBOARD_F11	
DIKEYBOARD_F12	
DIKEYBOARD_F13	On NEC PC-98 Japanese keyboard
DIKEYBOARD_F14	On NEC PC-98 Japanese keyboard
DIKEYBOARD_F15	On NEC PC-98 Japanese keyboard
DIKEYBOARD_G	
DIKEYBOARD_GRAVE	Grave accent (`)
DIKEYBOARD_H	
DIKEYBOARD_HOME	
DIKEYBOARD_I	
DIKEYBOARD_INSERT	
DIKEYBOARD_J	
DIKEYBOARD_K	
DIKEYBOARD_KANA	On Japanese keyboard
DIKEYBOARD_KANJI	On Japanese keyboard

DIKEYBOARD_L	
DIKEYBOARD_LBRACKET	Left square bracket [
DIKEYBOARD_LCONTROL	Left CTRL
DIKEYBOARD_LEFT	LEFT ARROW
DIKEYBOARD_LMENU	Left ALT
DIKEYBOARD_LSHIFT	Left SHIFT
DIKEYBOARD_LWIN	Left Microsoft® Windows® logo key
DIKEYBOARD_M	
DIKEYBOARD_MAIL	
DIKEYBOARD_MEDIASELECT	Media Select key, which displays a selection of supported media players on the system
DIKEYBOARD_MEDIASTOP	
DIKEYBOARD_MINUS	On main keyboard
DIKEYBOARD_MULTIPLY	Asterisk (*) on numeric keypad
DIKEYBOARD_MUTE	
DIKEYBOARD_MYCOMPUTER	
DIKEYBOARD_N	
DIKEYBOARD_NEXT	PAGE DOWN
DIKEYBOARD_NEXTTRACK	Next track
DIKEYBOARD_NOCONVERT	On Japanese keyboard
DIKEYBOARD_NUMLOCK	
DIKEYBOARD_NUMPAD0	
DIKEYBOARD_NUMPAD1	
DIKEYBOARD_NUMPAD2	
DIKEYBOARD_NUMPAD3	
DIKEYBOARD_NUMPAD4	
DIKEYBOARD_NUMPAD5	
DIKEYBOARD_NUMPAD6	
DIKEYBOARD_NUMPAD7	
DIKEYBOARD_NUMPAD8	
DIKEYBOARD_NUMPAD9	
DIKEYBOARD_NUMPADCOMMA	On numeric keypad of NEC PC-98 Japanese keyboard
DIKEYBOARD_NUMPADENTER	
DIKEYBOARD_NUMPADEQUALS	On numeric keypad of NEC PC-98 Japanese keyboard
DIKEYBOARD_O	
DIKEYBOARD_OEM_102	On British and German keyboards

DIKEYBOARD_P	
DIKEYBOARD_PAUSE	
DIKEYBOARD_PERIOD	On main keyboard
DIKEYBOARD_PLAYPAUSE	
DIKEYBOARD_POWER	
DIKEYBOARD_PREVTRACK	Previous track; circumflex on Japanese keyboard
DIKEYBOARD_PRIOR	PAGE UP
DIKEYBOARD_Q	
DIKEYBOARD_R	
DIKEYBOARD_RBRACKET	Right square bracket]
DIKEYBOARD_RCONTROL	Right CTRL
DIKEYBOARD_RETURN	ENTER on main keyboard
DIKEYBOARD_RIGHT	RIGHT ARROW
DIKEYBOARD_RMENU	Right ALT
DIKEYBOARD_RSHIFT	Right SHIFT
DIKEYBOARD_RWIN	Right Windows logo key
DIKEYBOARD_S	
DIKEYBOARD_SCROLL	SCROLL LOCK
DIKEYBOARD_SEMICOLON	
DIKEYBOARD_SLASH	Forward slash (/) on main keyboard
DIKEYBOARD_SLEEP	
DIKEYBOARD_SPACE	SPACEBAR
DIKEYBOARD_STOP	On NEC PC-98 Japanese keyboard
DIKEYBOARD_SUBTRACT	MINUS SIGN (-) on numeric keypad
DIKEYBOARD_SYSRQ	
DIKEYBOARD_T	
DIKEYBOARD_TAB	
DIKEYBOARD_U	
DIKEYBOARD_UNDERLINE	On NEC PC-98 Japanese keyboard
DIKEYBOARD_UNLABELED	On Japanese keyboard
DIKEYBOARD_UP	UP ARROW
DIKEYBOARD_V	
DIKEYBOARD_VOLUMEDOWN	
DIKEYBOARD_VOLUMEUP	
DIKEYBOARD_W	
DIKEYBOARD_WAKE	
DIKEYBOARD_WEBBACK	

DIKEYBOARD_WEBFAVORITES	Displays the Microsoft Internet Explorer Favorites list, the Windows Favorites folder, or the Netscape Bookmarks list.
DIKEYBOARD_WEBFORWARD	
DIKEYBOARD_WEBHOME	
DIKEYBOARD_WEBREFRESH	
DIKEYBOARD_WEBSEARCH	
DIKEYBOARD_WEBSTOP	
DIKEYBOARD_X	
DIKEYBOARD_Y	
DIKEYBOARD_YEN	On Japanese keyboard
DIKEYBOARD_Z	

Mouse Mapping Constants

The following constants are used in the **dwSemantic** member of the **DIACTION** structure to map an action to a physical axis or button on a mouse.

DIMOUSE_BUTTON0
DIMOUSE_BUTTON1
DIMOUSE_BUTTON2
DIMOUSE_BUTTON3
DIMOUSE_BUTTON4
DIMOUSE_BUTTON5
DIMOUSE_BUTTON6
DIMOUSE_BUTTON7
DIMOUSE_WHEEL
DIMOUSE_XAXIS
DIMOUSE_YAXIS
DIMOUSE_XAXISAB
DIMOUSE_YAXISAB

Note

DIMOUSE_XAXISAB and DIMOUSE_YAXISAB represent the x-axis and y-axis on a mouse that returns absolute rather than relative axis data.

DirectPlay Voice Mapping Constants

The following constants are used in the **dwSemantic** member of the **DIACTION** structure to map an action to a channel on Microsoft® a DirectPlay® voice device.

DIVOICE_CHANNEL1
DIVOICE_CHANNEL2
DIVOICE_CHANNEL3
DIVOICE_CHANNEL4
DIVOICE_CHANNEL5
DIVOICE_CHANNEL6
DIVOICE_CHANNEL7
DIVOICE_CHANNEL8
DIVOICE_ALL
DIVOICE_TEAM
DIVOICE_PLAYBACKMUTE
DIVOICE_RECORDMUTE
DIVOICE_TRANSMIT
DIVOICE_VOICECOMMAND

Any-Control Constants

The following constants are used in the **dwSemantic** member of the **DIACTION** structure to map an action to any matching control on the device.

Constant	Device object
DIAXIS_ANY_1	Any axis
DIAXIS_ANY_2	Any axis
DIAXIS_ANY_3	Any axis
DIAXIS_ANY_4	Any axis
DIAXIS_ANY_A_1	Any accelerator
DIAXIS_ANY_A_2	Any accelerator
DIAXIS_ANY_B_1	Any brake
DIAXIS_ANY_B_2	Any brake
DIAXIS_ANY_C_1	Any clutch
DIAXIS_ANY_C_2	Any clutch
DIAXIS_ANY_R_1	Any r-axis
DIAXIS_ANY_R_2	Any r-axis

DIAXIS_ANY_S_1	Any s-axis
DIAXIS_ANY_S_2	Any s-axis
DIAXIS_ANY_U_1	Any u-axis
DIAXIS_ANY_U_2	Any u-axis
DIAXIS_ANY_V_1	Any v-axis
DIAXIS_ANY_V_2	Any v-axis
DIAXIS_ANY_X_1	Any x-axis
DIAXIS_ANY_X_2	Any x-axis
DIAXIS_ANY_Y_1	Any y-axis
DIAXIS_ANY_Y_2	Any y-axis
DIAXIS_ANY_Z_1	Any z-axis
DIAXIS_ANY_Z_2	Any z-axis
DIBUTTON_ANY(x)	Any button
DIPOV_ANY_1	Any point-of-view controller
DIPOV_ANY_2	Any point-of-view controller
DIPOV_ANY_3	Any point-of-view controller
DIPOV_ANY_4	Any point-of-view controller

These constants can be used to map an application action to a virtual control that is not defined in a genre. Such actions are mapped after genre-specific actions. If the mapper has already mapped all matching controls to genre-specific actions, the any-control action is left unmapped.

Any-control actions are treated with equal priority. If a device has one x-axis and the action array specifies DIAXIS_ANY_1 and DIAXIS_ANY_X_1, the action mapped to the x-axis is the one that appears first in the action array.

DIBUTTON_ANY(x) can accept any value from 0 through 255. Do not use any given index more than once. For instance, the following set of **DIACTION** structures would be invalid.

```
{eB_MUTE,    DIBUTTON_ANY(0), 0, "Toggle Sound",},
{eB_VOLUP,   DIBUTTON_ANY(0), 0, "Volume Up",   },
{eB_VOLDOWN, DIBUTTON_ANY(0), 0, "Volume Down", },
```

The correct method would use different indexes as shown below.

```
{eB_MUTE,    DIBUTTON_ANY(0), 0, "Toggle Sound",}
{eB_VOLUP,   DIBUTTON_ANY(1), 0, "Volume Up",   }
{eB_VOLDOWN, DIBUTTON_ANY(2), 0, "Volume Down", }
```

Return Values

The list below contains the **HRESULT** values that can be returned by Microsoft® DirectInput® methods and functions. Errors are represented by negative values and cannot be combined.

For a list of the error values each method or function can return, see the individual descriptions. Lists of error codes in the documentation are necessarily incomplete. For example, any DirectInput method can return DIERR_OUTOFMEMORY even though the error code is not explicitly listed as a possible return value in the documentation for that method.

DI_BUFFEROVERFLOW
: The device buffer overflowed and some input was lost. This value is equal to the S_FALSE standard COM return value.

DI_DOWNLOADSKIPPED
: The parameters of the effect were successfully updated, but the effect could not be downloaded because the associated device was not acquired in exclusive mode.

DI_EFFECTRESTARTED
: The effect was stopped, the parameters were updated, and the effect was restarted.

DI_NOEFFECT
: The operation had no effect. This value is equal to the S_FALSE standard COM return value.

DI_NOTATTACHED
: The device exists but is not currently attached. This value is equal to the S_FALSE standard COM return value.

DI_OK
: The operation completed successfully. This value is equal to the S_OK standard COM return value.

DI_POLLEDDEVICE
: The device is a polled device. As a result, device buffering does not collect any data and event notifications is not signaled until the **IDirectInputDevice8::Poll** method is called.

DI_PROPNOEFFECT
: The change in device properties had no effect. This value is equal to the S_FALSE standard COM return value.

DI_SETTINGSNOTSAVED
: The action map was applied to the device, but the settings could not be saved.

DI_TRUNCATED
: The parameters of the effect were successfully updated, but some of them were beyond the capabilities of the device and were truncated to the nearest supported value.

DI_TRUNCATEDANDRESTARTED
: Equal to DI_EFFECTRESTARTED | DI_TRUNCATED.

DI_WRITEPROTECT
A SUCCESS code indicating that settings cannot be modified.

DIERR_ACQUIRED
The operation cannot be performed while the device is acquired.

DIERR_ALREADYINITIALIZED
This object is already initialized

DIERR_BADDRIVERVER
The object could not be created due to an incompatible driver version or mismatched or incomplete driver components.

DIERR_BETADIRECTINPUTVERSION
The application was written for an unsupported prerelease version of DirectInput.

DIERR_DEVICEFULL
The device is full.

DIERR_DEVICENOTREG
The device or device instance is not registered with DirectInput. This value is equal to the REGDB_E_CLASSNOTREG standard COM return value.

DIERR_EFFECTPLAYING
The parameters were updated in memory but were not downloaded to the device because the device does not support updating an effect while it is still playing.

DIERR_GENERIC
An undetermined error occurred inside the DirectInput subsystem. This value is equal to the E_FAIL standard COM return value.

DIERR_HANDLEEXISTS
The device already has an event notification associated with it. This value is equal to the E_ACCESSDENIED standard COM return value.

DIERR_HASEFFECTS
The device cannot be reinitialized because effects are attached to it.

DIERR_INCOMPLETEEFFECT
The effect could not be downloaded because essential information is missing. For example, no axes have been associated with the effect, or no type-specific information has been supplied.

DIERR_INPUTLOST
Access to the input device has been lost. It must be reacquired.

DIERR_INVALIDPARAM
An invalid parameter was passed to the returning function, or the object was not in a state that permitted the function to be called. This value is equal to the E_INVALIDARG standard COM return value.

DIERR_MAPFILEFAIL
An error has occured either reading the vendor-supplied action-mapping file for the device or reading or writing the user configuration mapping file for the device.

DIERR_MOREDATA
Not all the requested information fit into the buffer.

DIERR_NOAGGREGATION
This object does not support aggregation.

DIERR_NOINTERFACE
The object does not support the specified interface. This value is equal to the E_NOINTERFACE standard COM return value.

DIERR_NOTACQUIRED
The operation cannot be performed unless the device is acquired.

DIERR_NOTBUFFERED
The device is not buffered. Set the DIPROP_BUFFERSIZE property to enable buffering.

DIERR_NOTDOWNLOADED
The effect is not downloaded.

DIERR_NOTEXCLUSIVEACQUIRED
The operation cannot be performed unless the device is acquired in DISCL_EXCLUSIVE mode.

DIERR_NOTFOUND
The requested object does not exist.

DIERR_NOTINITIALIZED
This object has not been initialized.

DIERR_OBJECTNOTFOUND
The requested object does not exist.

DIERR_OLDDIRECTINPUTVERSION
The application requires a newer version of DirectInput.

DIERR_OTHERAPPHASPRIO
Another application has a higher priority level, preventing this call from succeeding. This value is equal to the E_ACCESSDENIED standard COM return value. This error can be returned when an application has only foreground access to a device but is attempting to acquire the device while in the background.

DIERR_OUTOFMEMORY
The DirectInput subsystem could not allocate sufficient memory to complete the call. This value is equal to the E_OUTOFMEMORY standard COM return value.

DIERR_READONLY
The specified property cannot be changed. This value is equal to the E_ACCESSDENIED standard COM return value.

DIERR_REPORTFULL
More information was requested to be sent than can be sent to the device.

DIERR_UNPLUGGED
The operation could not be completed because the device is not plugged in.

DIERR_UNSUPPORTED
The function called is not supported at this time. This value is equal to the E_NOTIMPL standard COM return value.

E_HANDLE

The HWND parameter is not a valid top-level window that belongs to the process.

E_PENDING
Data is not yet available.

E_POINTER
An invalid pointer, usually NULL, was passed as a parameter.

DirectInput Visual Basic Reference

Reference material for the Microsoft® DirectInput® Microsoft Visual Basic® application programming interface (API) is divided into the following categories.

- Classes
- Types
- Enumerations
- Keyboard Keys
- Action Mapping Constants
- Error Codes

Classes

This section contains references for methods of the following Microsoft® DirectInput® classes.

- **DirectInput8**
- **DirectInputDevice8**
- **DirectInputDeviceInstance8**
- **DirectInputDeviceObjectInstance**
- **DirectInputEffect**
- **DirectInputEnumDeviceObjects**
- **DirectInputEnumDevices8**
- **DirectInputEnumEffects**

DirectInput8

#The **DirectInput8** class represents the Microsoft® DirectInput® system. An application should have a single object of this class, which is used to enumerate

IDH_DirectInput8_dinput_vb

available devices, create devices, and retrieve the status of devices, as well as to invoke an instance of the Microsoft Windows® Control Panel.

The **DirectInput8** object is obtained by using the **DirectX8.DirectInputCreate** method.

The **DirectInput8** class has the following methods:

Action Mapping	**ConfigureDevices**
	GetDevicesBySemantics
Device Management	**CreateDevice**
	GetDeviceStatus
	GetDIDevices
Miscellaneous	**RunControlPanel**

DirectInput8.ConfigureDevices

#Displays property pages for connected input devices and enables the user to map actions to device controls.

object.**ConfigureDevices(** _
hEvent **As Long,** _
CDParams **As DICONFIGUREDEVICESPARAMS,** _
flags **As Long)**

Parts

object
Resolves to a **DirectInput8** object.

hEvent
Handle to an event that is set each time a change is made to the display of the property sheet. Can be 0 or a value returned by **DirectX8.CreateEvent**.

CDParams
DICONFIGUREDEVICESPARAMS type that contains parameters for action mapping.

flags
Value that specifies the mode in which the control panel should be invoked. Must be one of the following values from the **CONST_DICDFLAGS** enumeration.

DICD_DEFAULT
Open the property sheet in view-only mode.

DICD_EDIT
Open the property sheet in edit mode. This mode enables the user to change action-to-control mappings. After the call returns, the application should

IDH_DirectInput8.ConfigureDevices_dinput_vb

assume current devices are no longer valid, release all device interfaces, and reinitialize them by calling **DirectInput8.GetDevicesBySemantics**.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_INVALIDPARAM
DIERR_OUTOFMEMORY

Remarks

Hardware vendors provide bitmaps and other display information for their device.

Before calling the method, an application can modify the text labels associated with each action by changing the value in the **ActionName** member of the **DIACTION** type.

Configuration is stored for each user of each device for each game. The information can be retrieved by the **DirectInputDevice8.BuildActionMap** method.

By default, acceleration is supported for these pixel formats:

A1R5G5B5
: 16-bit pixel format with 5 bits reserved for each color and 1 bit reserved for alpha (transparent texel).

A8R8G8B8
: 32-bit ARGB pixel format with alpha.

R9G8B8
: 24-bit RGB pixel format.

X1R5G5B5
: 16-bit pixel format with 5 bits reserved for each color.

X8R8G8B8
: 32-bit RGB pixel format with 8 bits reserved for each color.

Other formats result in color conversion and dramatically slow the frame rate.

Note

Even if the cooperative level for the application is disabling the Windows logo key passively through an exclusive cooperative level or actively through use of the DISCL_NOWINKEY flag, that key is active while the default action mapping UI is displayed.

DirectInput8.CreateDevice

#Creates and initializes an instance of a device based on a given GUID.

IDH_DirectInput8.CreateDevice_dinput_vb

object.**CreateDevice(***guid* **As String) As DirectInputDevice8**

Parts

object
Resolves to a **DirectInput8** object.

guid
The instance GUID for the desired input device. The GUID is retrieved from the **DirectInputDeviceInstance8** object returned by the **DirectInputEnumDevices8.GetItem** method, or it can be one of the following strings:

GUID_SysKeyboard
The default system keyboard.

GUID_SysMouse
The default system mouse.

Return Values

Returns a **DirectInputDevice8** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_DEVICENOTREG
DIERR_INVALIDPARAM
DIERR_NOINTERFACE
DIERR_OUTOFMEMORY

See Also

DirectInput8, Using GUIDs

DirectInput8.GetDevicesBySemantics

#Enumerates devices that most closely match the application-specified action map.

```
object.GetDevicesBySemantics( _
  str1 As String, _
  format As DIACTIONFORMAT, _
  flags As Long _
) As DirectInputEnumDevices8
```

IDH_DirectInput8.GetDevicesBySemantics_dinput_vb

Parts

object
Resolves to a **DirectInput8** object.

str1
String identifying the current user, or *vbNullString* to specify the user logged onto the system. The user name is taken into account when enumerating devices. A device with user mappings is preferred to a device without any user mappings. By default, devices in use by other users are not enumerated for this user.

format
DIACTIONFORMAT type that specifies the action map for which suitable devices are enumerated.

flags
Flag value that specifies the scope of the enumeration. Can be one or more values from the **CONST_DIEDBSFLFLAGS** enumeration.

Return Values

Returns a **DirectInputEnumDevices8** object that represents the collection of enumerated devices.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

The keyboard and mouse are enumerated last.

See Also

Action Mapping

DirectInput8.GetDeviceStatus

#Determines whether a device is attached to the system.

object.**GetDeviceStatus(***guid* **As String) As Boolean**

Parts

object
Resolves to a **DirectInput8** object.

guid

IDH_DirectInput8.GetDeviceStatus_dinput_vb

The instance GUID for the desired input device. The GUID is retrieved by using the **DirectInputDeviceInstance8.GetGuidInstance** method on the object returned by **DirectInputEnumDevices8.GetItem** method, or it can be one of the following strings:

GUID_SysKeyboard
The default system keyboard.

GUID_SysMouse
The default system mouse.

Return Values

Returns True if the device is attached, and False otherwise.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_GENERIC

DIERR_INVALIDPARAM

See Also

DirectInput8, Using GUIDs

DirectInput8.GetDIDevices

#Enumerates available devices.

```
object.GetDIDevices( _
  deviceType As CONST_DI8DEVICETYPE, _
  flags As CONST_DIENUMDEVICESFLAGS) _
  As DirectInputEnumDevices8
```

Parts

object
Resolves to a **DirectInput8** object.

deviceType
Value that specifies the type of device to enumerate. If this parameter is 0, all types are enumerated. Otherwise, it is one of the constants of the **CONST_DI8DEVICETYPE** enumeration representing a class or type of device.

flags

IDH_DirectInput8.GetDIDevices_dinput_vb

Flag value that specifies the scope of the enumeration. This parameter contains one or more of the constants of the **CONST_DIENUMDEVICESFLAGS** enumeration.

Return Values

Returns a **DirectInputEnumDevices8** object.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

All installed devices can be enumerated, even if they are not present. For example, a joystick may be installed on the system but not currently plugged into the computer.

If a single piece of hardware can function as more than one Microsoft® DirectInput® device type, it will be returned for each device type it supports. For example, a keyboard with a built-in mouse will be enumerated as a keyboard and as a mouse. The product GUID would be the same for each device, however.

See Also

DirectInput8

DirectInput8.RunControlPanel

#Runs the Microsoft® Windows® Control Panel to enable the user to install a new input device or modify configurations.

object.**RunControlPanel(***hwndOwner* **As Long)**

Parts

object
Resolves to a **DirectInput8** object.

hwndOwner
Handle to the window to be used as the parent window for the subsequent user interface. If this parameter is 0, no parent window is used.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

IDH_DirectInput8.RunControlPanel_dinput_vb

See Also

DirectInput8, **DirectInputDevice8.RunControlPanel**

DirectInputDevice8

#Applications use the methods of the **DirectInputDevice8** class to gain and release access to Microsoft® DirectInput® devices, manage device properties and information, set behavior, perform initialization, and invoke a device's property sheet.

The **DirectInputDevice8** object is obtained by using the **DirectInput8.CreateDevice** method.

The methods of the **DirectInputDevice8** class can be organized into the following groups.

Access	**Acquire**
	SetCooperativeLevel
	Unacquire
Action Mapping	**BuildActionMap**
	GetImageInfo
	GetImageInfoCount
	SetActionMap
Data Retrieval	**GetDeviceData**
	GetDeviceState
	GetDeviceStateJoystick
	GetDeviceStateJoystick2
	GetDeviceStateKeyboard
	GetDeviceStateMouse
	GetDeviceStateMouse2
	Poll
	SetEventNotification
Force Feedback	**CreateCustomEffect**
	CreateEffect
	CreateEffectFromFile
	GetEffectsEnum
	GetForceFeedbackState
	SendForceFeedbackCommand
	WriteEffectToFile
Objects	**GetDeviceObjectsEnum**

IDH_DirectInputDevice8_dinput_vb

	GetObjectInfo
Properties	**GetCapabilities**
	GetDeviceInfo
	GetProperty
	SetCommonDataFormat
	SetDataFormat
	SetProperty
Miscellaneous	**RunControlPanel**
	SendDeviceData

DirectInputDevice8.Acquire

#Obtains access to the input device.

object.**Acquire()**

Parts

object
Resolves to a **DirectInputDevice8** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.:

DIERR_INVALIDPARAM
DIERR_OTHERAPPHASPRIO

Remarks

Before a device can be acquired, a data format must be set by using the **DirectInputDevice8.SetDataFormat** or **DirectInputDevice8.SetCommonDataFormat** method.

A device must be acquired before input data can be retrieved from it.

See Also

DirectInputDevice8.Unacquire

DirectInputDevice8.BuildActionMap

#Builds an action map for the device and retrieves information about it.

IDH_DirectInputDevice8.Acquire_dinput_vb
IDH_DirectInputDevice8.BuildActionMap_dinput_vb

object.**BuildActionMap(** _
format **As DIACTIONFORMAT,** _
username **As String,** _
flags **As Long)**

Parts

object
Resolves to a **DirectInputDevice8** object.

format
DIACTIONFORMAT type that receives properties of the action map.

username
String that specifies the name of the user for whom mapping is requested. If an empty string, the current user is assumed.

flags
Flags used to control the mapping. Can be one of the constants of the **CONST_DIDBAMFLAGS** enumeration.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM
DIERR_MAPFILEFAIL

Remarks

If DIERR_INVALIDPARAM is raised, one or more of the mappings was not valid. The **lHow** member of the **DIACTION** type is set to DIAH_ERROR. The application can iterate through the action map to find and correct errors.

See Also

DirectInputDevice8.SetActionMap, **DirectInputDevice8.SetDataFormat**

DirectInputDevice8.CreateCustomEffect

#Creates a force-feedback effect consisting of a series of constant forces of fixed duration.

object.**CreateCustomEffect(***effectinfo* **As DIEFFECT**, _
channels **As Long**, *samplePeriod* **As Long**, _
nSamples **As Long**, *sampledata()* **As Long)** _

IDH_DirectInputDevice8.CreateCustomEffect_dinput_vb

As DirectInputEffect

Parts

object
Resolves to a **DirectInputDevice8** object.

effectinfo
DIEFFECT type containing general parameters of the effect.

channels
The number of channels (axes) affected by this force. Must be 1 or 2.

If there is only a single channel, then the effect will be rotated in the direction specified by the **x** member of the **DIEFFECT** type. Not all devices support rotation of custom effects.

If there is more than one channel, the first channel is applied to the x-axis and the second to the y-axis. Rotation is not allowed.

samplePeriod
The sample period in microseconds. See Remarks.

nSamples
Number of elements in the *sampledata* array.

sampledata
Array of magnitudes. If *channels* is greater than 1, then the values are interleaved. For example, if *channels* is 2, then the first element of the array is assigned to the x-axis, the second to the y-axis, the third to the x-axis, and so on.

Remarks

In theory, *samplePeriod* is the length of time for which each magnitude in *sampledata* is valid, whereas **DIEFFECT.lSamplePeriod** is the length of time between samplings of the data (and the minimum time between changes in magnitude). Since each element in the array needs to be sampled exactly once on each iteration through the array, and some drivers ignore *samplePeriod* in any case, it is best to make the values of **lSamplePeriod** and *samplePeriod* identical.

See Also

Custom Forces

DirectInputDevice8.CreateEffect

#Creates a force-feedback effect. If the device is currently acquired at the exclusive cooperative level, the effect is also downloaded.

object.**CreateEffect(***effectGuid* **As String, _**
effectinfo **As DIEFFECT) As DirectInputEffect**

IDH_DirectInputDevice8.CreateEffect_dinput_vb

Parts

object
Resolves to a **DirectInputDevice8** object.

effectGuid
String representation of a GUID for an effect recognized by the hardware driver, or one of the following aliases for standard effects:

GUID_ConstantForce
GUID_RampForce
GUID_Square
GUID_Sine
GUID_Triangle
GUID_SawtoothUp
GUID_SawtoothDown
GUID_Spring
GUID_Damper
GUID_Inertia
GUID_Friction

effectinfo
DIEFFECT type containing the parameters for the effect.

Return Values

Returns a **DirectInputEffect** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_DEVICEFULL
DIERR_DEVICENOTREG
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

If no error is raised, the effect was created and the parameters of the effect were updated, but the effect was not necessarily downloaded. In order for it to be downloaded, the device must be acquired in exclusive mode.

See Also

DirectInputDevice8.CreateEffectFromFile

DirectInputEffect.Download

DirectInputEffect.Start

DirectInputDevice8.CreateEffectFromFile

#Creates a force-feedback effect whose parameters are stored in a file. If the device is currently acquired at the exclusive cooperative level, the effect is also downloaded.

```
object.CreateEffectFromFile( _
  filename As String, _
  flags As Long, _
  effectName As String _
) As DirectInputEffect
```

Parts

object
Resolves to a **DirectInputDevice8** object.

filename
String that specifies the file name.

flags
Flags that specify how to create the effect. Can be one or more values from the **CONST_DIFEFFLAGS** enumeration.

effectName
Friendly name of the effect, assigned by the author.

Return Values

Returns a **DirectInputEffect** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_DEVICEFULL
DIERR_DEVICENOTREG
DIERR_INVALIDPARAM
DIERR_NOTINITIALIZED

Remarks

If no error is raised, the effect was created but not necessarily downloaded. In order for it to be downloaded, the device must be acquired in exclusive mode.

IDH_DirectInputDevice8.CreateEffectFromFile_dinput_vb

Effect files must be in the format used by the Force Editor (Fedit.exe) application supplied with the Microsoft® DirectInput® SDK.

See Also

DirectInputEffect.Download, **DirectInputDevice8.WriteEffectToFile**

DirectInputDevice8.GetCapabilities

#Retrieves the capabilities of the device.

object.**GetCapabilities(***caps* **As DIDEVCAPS)**

Parts

object
Resolves to a **DirectInputDevice8** object.

caps
A **DIDEVCAPS** type to be filled with the device capabilities.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDevice8.GetDeviceData

#Retrieves buffered data from the device.

object.**GetDeviceData(_**
deviceObjectDataArray() **As DIDEVICEOBJECTDATA, _**
flags **As CONST_DIDGDDFLAGS) As Long**

Parts

object
Resolves to a **DirectInputDevice8** object.

deviceObjectDataArray()
Array of **DIDEVICEOBJECTDATA** types to receive the buffered data.

flags
Flags that control the manner in which data is obtained. This value can be 0 or one of the constants of the **CONST_DIDGDDFLAGS** enumeration.

Return Values

Returns the number of buffered data elements actually returned in *deviceObjectDataArray*.

IDH_DirectInputDevice8.GetCapabilities_dinput_vb
IDH_DirectInputDevice8.GetDeviceData_dinput_vb

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
DIERR_NOTBUFFERED

Remarks

Before device data can be obtained, you must set the data format by using the **DirectInputDevice8.SetDataFormat** method, set the buffer size by using **DirectInputDevice8.SetProperty** method, and acquire the device by using the **DirectInputDevice8.Acquire** method.

You can use this method to retrieve one or more input events from the buffer you created by using **SetProperty**. You do not have to retrieve all pending events with a single call. You can, for example, pass in a *deviceObjectDataArray()* consisting of a single element and loop on **GetDeviceData** until no more data is returned.

If the buffer overflows, all pending data is lost and the DI_BUFFEROVERFLOW error is raised.

See Also

DirectInputDevice8.Poll, Polling and Event Notification

DirectInputDevice8.GetDeviceInfo

#Retrieves information about the device's identity.

object.**GetDeviceInfo() As DirectInputDeviceInstance8**

Parts

object
Resolves to a **DirectInputDevice8** object.

Return Values

Returns a **DirectInputDeviceInstance8** object

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

IDH_DirectInputDevice8.GetDeviceInfo_dinput_vb

DirectInputDevice8.GetDeviceObjectsEnum

[#]Enumerates the input and output objects available on a device.

```
object.GetDeviceObjectsEnum( _
  flags As CONST_DIDFTFLAGS) _
  As DirectInputEnumDeviceObjects
```

Parts

object
Resolves to a **DirectInputDevice8** object.

flags
Flags specifying the type of object to be enumerated. Can be one or more of the members of the **CONST_DIDFTFLAGS** enumeration.

Return Values

If the method succeeds, the return value is a **DirectInputEnumDeviceObjects** object that represents the collection of enumerated devices.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

The DIDFT_FFACTUATOR and DIDFT_FFEFFECTTRIGGER flags restrict enumeration to objects that meet all the criteria defined by the included flags. For all the other flags, an object is enumerated if it meets the criterion defined by any included flag in this category. For example, (DIDFT_FFACTUATOR **Or** DIDFT_FFEFFECTTRIGGER) restricts enumeration to force-feedback trigger objects, and (DIDFT_FFEFFECTTRIGGER **Or** DIDFT_TGLBUTTON **Or** DIDFT_PSHBUTTON) restricts enumeration to buttons of any kind that can be used as effect triggers.

Applications should not rely on enumeration to determine whether certain keyboard keys or indicator lights are present, as these objects might be enumerated even though they are not present. Although the basic set of available objects can be determined from the device subtype, there is no reliable way of determining whether extra objects such as the menu key are available.

[#] IDH_DirectInputDevice8.GetDeviceObjectsEnum_dinput_vb

DirectInputDevice8.GetDeviceState

#Retrieves immediate data for a device other than a standard keyboard, mouse, or joystick.

object.**GetDeviceState(***cb* **As Long,** *state* **As Any)**

Parts

object
Resolves to a **DirectInputDevice8** object.

cb
Size of the array whose first element is passed as *state*.

state
First element of an array to receive device state information.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
E_PENDING

Remarks

Before device data can be obtained, you must set the cooperative level by using the **DirectInputDevice8.SetCooperativeLevel** method, then set the data format by using **DirectInputDevice8.SetDataFormat**, and acquire the device by using the **DirectInputDevice8.Acquire** method.

See Also

DirectInputDevice8.GetDeviceStateJoystick, **DirectInputDevice8.GetDeviceStateJoystick2**, **DirectInputDevice8.GetDeviceStateKeyboard,** **DirectInputDevice8.GetDeviceStateMouse**, **DirectInputDevice8.SetDataFormat**, Buffered and Immediate Data

IDH_DirectInputDevice8.GetDeviceState_dinput_vb

DirectInputDevice8.GetDeviceStateJoystick

#Retrieves immediate data from a joystick device.

object.**GetDeviceStateJoystick(***state* **As DIJOYSTATE)**

Parts

object
Resolves to a **DirectInputDevice8** object.

state
A **DIJOYSTATE** type that receives the current state of the device.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
E_PENDING

Remarks

Before device data can be obtained, you must set the cooperative level by using the **DirectInputDevice8.SetCooperativeLevel** method, then set the data format by using **DirectInputDevice8.SetCommonDataFormat**, and acquire the device by using the **DirectInputDevice8.Acquire** method.

See Also

DirectInputDevice8.Poll Polling and Event Notification

DirectInputDevice8.GetDeviceStateJoystick2

#Retrieves immediate data from a joystick device with extended capabilities.

object.**GetDeviceStateJoystick2(***state* **As DIJOYSTATE2)**

IDH_DirectInputDevice8.GetDeviceStateJoystick_dinput_vb
IDH_DirectInputDevice8.GetDeviceStateJoystick2_dinput_vb

Parts

object
Resolves to a **DirectInputDevice8** object.

state
A **DIJOYSTATE2** type that receives the current state of the device. The format of the data is established by a prior call to the **DirectInputDevice8.SetDataFormat** method.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
E_PENDING

Remarks

Before device data can be obtained, you must set the cooperative level by using the **DirectInputDevice8.SetCooperativeLevel** method, then set the data format by using **DirectInputDevice8.SetCommonDataFormat**, and acquire the device by using the **DirectInputDevice8.Acquire** method.

See Also

DirectInputDevice8.Poll, Polling and Event Notification

DirectInputDevice8.GetDeviceStateKeyboard

#Retrieves immediate data from a keyboard device.

object.**GetDeviceStateKeyboard(***state* **As DIKEYBOARDSTATE)**

Parts

object
Resolves to a **DirectInputDevice8** object.

state
A **DIKEYBOARDSTATE** type that receives the current state of the device.

IDH_DirectInputDevice8.GetDeviceStateKeyboard_dinput_vb

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
E_PENDING

Remarks

Before device data can be obtained, you must set the cooperative level by using the **DirectInputDevice8.SetCooperativeLevel** method, then set the data format by using **DirectInputDevice8.SetCommonDataFormat**, and acquire the device by using the **DirectInputDevice8.Acquire** method.

See Also

DirectInputDevice8.Poll, Polling and Event Notification

DirectInputDevice8.GetDeviceStateMouse

#Retrieves immediate data from a mouse device that has up to four buttons.

object.**GetDeviceStateMouse(***state* **As DIMOUSESTATE)**

Parts

object
Resolves to a **DirectInputDevice8** object.

state
A **DIMOUSESTATE** type that receives the state of the device.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
E_PENDING

IDH_DirectInputDevice8.GetDeviceStateMouse_dinput_vb

Remarks

Before device data can be obtained, you must set the cooperative level by using the **DirectInputDevice8.SetCooperativeLevel** method, then set the data format by using **DirectInputDevice8.SetCommonDataFormat**, and acquire the device by using the **DirectInputDevice8.Acquire** method.

See Also

DirectInputDevice8.GetDeviceStateMouse2, **DirectInputDevice8.Poll** Polling and Event Notification

DirectInputDevice8.GetDeviceStateMouse2

#Retrieves immediate data from a mouse device that has up to eight buttons.

object.**GetDeviceStateMouse2(***state* **As DIMOUSESTATE2)**

Parts

object
Resolves to a **DirectInputDevice8** object.

state
A **DIMOUSESTATE2** type that receives the state of the device.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTACQUIRED
E_PENDING

Remarks

Before device data can be obtained, you must set the cooperative level by using the **DirectInputDevice8.SetCooperativeLevel** method, then set the data format by using **DirectInputDevice8.SetCommonDataFormat**, and acquire the device by using the **DirectInputDevice8.Acquire** method.

IDH_DirectInputDevice8.GetDeviceStateMouse2_dinput_vb

See Also

DirectInputDevice8.GetDeviceStateMouse, **DirectInputDevice8.Poll**, Polling and Event Notification

DirectInputDevice8.GetEffectsEnum

#Enumerates force-feedback effects supported by the device, including standard effects as well as effects designed by the device manufacturer.

```
object.GetEffectsEnum( _
  effType As CONST_DIEFTFLAGS _
) As DirectInputEnumEffects
```

Parts

object
Resolves to a **DirectInputDevice8** object.

effType
One of the following flags from the **CONST_DIEFTFLAGS** enumeration specifying the type of effect to be enumerated:

DIEFT_ALL
DIEFT_CONDITION
DIEFT_CONSTANTFORCE
DIEFT_CUSTOMFORCE
DIEFT_HARDWARE
DIEFT_PERIODIC
DIEFT_RAMPFORCE

Return Values

Returns a **DirectInputEnumEffects** object whose methods can be used to retrieve information about the effects.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDevice8.GetForceFeedbackState

#Retrieves the state of the device's force-feedback system.

```
object.GetForceFeedbackState() As CONST_DIGFFSFLAGS
```

IDH_DirectInputDevice8.GetEffectsEnum_dinput_vb
IDH_DirectInputDevice8.GetForceFeedbackState_dinput_vb

Parts

object
Resolves to a **DirectInputDevice8** object.

Return Values

Returns flags from the **CONST_DIGFFSFLAGS** enumeration that describe the current state of the device's force-feedback system.

Future versions of Microsoft® DirectInput® may define additional flags. Applications should ignore any flags that are not currently defined.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED
DIERR_UNSUPPORTED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

DirectInputDevice8.GetImageInfo

#Retrieves information about a device image for use in a configuration property sheet.

*object.***GetImageInfo(***info* **As DIDEVICEIMAGEINFOHEADER)**

Parts

object
Resolves to a **DirectInputDevice8** object.

info
DIDEVICEIMAGEINFOHEADER type that receives information about the device image.

IDH_DirectInputDevice8.GetImageInfo_dinput_vb

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM
DIERR_MAPFILEFAIL
DIERR_MOREDATA
DIERR_NOTINITIALIZED
DIERR_OBJECTNOTFOUND

See Also

DirectInputDevice8.GetImageInfoCount

DirectInputDevice8.GetImageInfoCount

#Retrieves the number of images available for the device. This method is used to ascertain how many array elements must be allocated in the **Images** member of the **DIDEVICEIMAGEINFOHEADER** type before the type is passed to the **DirectInputDevice8.GetImageInfo**.

object.**GetImageInfoCount() As Long**

Parts

object
Resolves to a **DirectInputDevice8** object.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDevice8.GetObjectInfo

#Retrieves information about a device object such as a button or axis.

object.**GetObjectInfo(_**
obj **As Long**, **_**
how **As CONST_DIPHFLAGS) _**
As DirectInputDeviceObjectInstance

Parts

object

IDH_DirectInputDevice8.GetImageInfoCount_dinput_vb
IDH_DirectInputDevice8.GetObjectInfo_dinput_vb

Resolves to a **DirectInputDevice8** object.

obj

Value that identifies the object whose information will be retrieved. The interpretation of this parameter depends on the value specified in the *how* parameter.

how

Value specifying how the *obj* parameter should be interpreted. This value can be one of the constants of the **CONST_DIPHFLAGS** enumeration.

Return Values

Returns a **DirectInputDeviceObjectInstance** object whose methods can be used to retrieve information about the object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM

DIERR_OBJECTNOTFOUND

DirectInputDevice8.GetProperty

#Retrieves information about the input device.

object.**GetProperty(** _
guid **As String**, _
propertyInfo **As Any)**

Parts

object

Resolves to a **DirectInputDevice8** object.

guid

Identifier of the property to be retrieved. The following properties are defined for an input device and can be passed as strings:

DIPROP_AUTOCENTER

Specifies whether device objects are self-centering. See **DirectInputDevice8.SetProperty** for more information.

DIPROP_AXISMODE

Retrieves the axis mode. The retrieved value can be DIPROPAXISMODE_ABS or DIPROPAXISMODE_REL. (See the **CONST_DINPUT** enumeration.)

DIPROP_BUFFERSIZE

IDH_DirectInputDevice8.GetProperty_dinput_vb

Retrieves the input-buffer size. The buffer size determines the amount of data that the buffer can hold between calls to the **DirectInputDevice8.GetDeviceData** method before data is lost. This value may be set to 0 to indicate that the application will not be reading buffered data from the device.

DIPROP_DEADZONE

Retrieves a value for the dead zone of a joystick, in the range 0 to 10,000, where 0 indicates there is no dead zone, 5,000 indicates that the dead zone extends over 50 percent of the physical range of the axis on both sides of center, and 10,000 indicates that the entire physical range of the axis is dead. When the axis is within the dead zone, it is reported as being at the center of its range.

DIPROP_FFGAIN

Retrieves the gain of the device. See **DirectInputDevice8.SetProperty** for more information.

DIPROP_FFLOAD

Retrieves the memory load for the device. This setting applies to the entire device, rather than to any particular object, so the **lHow** member of the associated **DIPROPLONG** type must be DIPH_DEVICE. The device must be acquired in exclusive mode. If it is not, the method will fail with a return value of DIERR_NOTEXCLUSIVEACQUIRED.

The **lData** member contains a value in the range 0 to 100, indicating the percentage of device memory in use.

DIPROP_GRANULARITY

Retrieves the input granularity. Granularity represents the smallest distance the object will report movement. Most axis objects have a granularity of 1, meaning that all values are possible. Some axes may have a larger granularity. For example, the wheel axis on a mouse may have a granularity of 20, meaning that all reported changes in position will be multiples of 20. In other words, when the user turns the wheel slowly, the device reports a position of 0, then 20, then 40, and so on.

This is a read-only property; you cannot set its value by calling the **DirectInputDevice8.SetProperty** method.

DIPROP_RANGE

Retrieves the range of values an object can possibly report. The retrieved minimum and maximum values are set in the **lMin** and **lMax** members of the associated **DIPROPRANGE** type.

For some devices, this is a read-only property; you cannot set its value by calling the **DirectInputDevice8.SetProperty** method.

DIPROP_SATURATION

Retrieves a value for the saturation zones of a joystick, in the range 0 to 10,000. The saturation level is the point at which the axis is considered to be at its most extreme position. For example, if the saturation level is set to 9,500, then the axis reaches the extreme of its range when it has moved 95

percent of the physical distance from its center position (or from the dead zone).

propertyInfo

A **DIPROPLONG** type to receive a single value, or a **DIPROPRANGE** type to receive a pair of values for the property. The **lObj**, **lHow**, and **lSize** members of the type must be initialized before the method is called.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED
DIERR_OBJECTNOTFOUND
DIERR_UNSUPPORTED

See Also

DirectInputDevice8.SetProperty

DirectInputDevice8.Poll

#Makes data available from polled objects on a Microsoft® DirectInput® device. If the device does not require polling, then calling this method has no effect. If a device that requires polling is not polled periodically, no new data will be received from the device. Calling this method causes Microsoft® DirectInput® to update the device state, generate input events (if buffered data is enabled), and set notification events (if notification is enabled).

object.**Poll()**

Parts

object

Resolves to a **DirectInputDevice8** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_NOTACQUIRED

IDH_DirectInputDevice8.Poll_dinput_vb

Remarks

Before a device data can be polled, the data format must be set by using the **DirectInputDevice8.SetDataFormat** or **DirectInputDevice8.SetCommonDataFormat** method, and the device must be acquired by using the **DirectInputDevice8.Acquire** method.

See Also

Polling and Event Notification

DirectInputDevice8.RunControlPanel

#Opens the Control Panel property sheet associated with this device. If the device does not have a property sheet associated with it, the default device property sheet is displayed.

object.**RunControlPanel(***hwnd* **As Long)**

Parts

object
Resolves to a **DirectInputDevice8** object.

hwnd
Handle to the parent window. If this parameter is 0, no parent window is used.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM

DirectInputDevice8.SendDeviceData

#Sends data to a device that accepts output. The device must be in an acquired state.

object.**SendDeviceData(***count* **As Long, _**
data() **As DIDEVICEOBJECTDATA, _**
flags **As CONST_DESDDFLAGS) As Long**

Parts

object

IDH_DirectInputDevice8.RunControlPanel_dinput_vb
IDH_DirectInputDevice8.SendDeviceData_dinput_vb

Resolves to a **DirectInputDevice8** object.

count

Number of elements in *data*.

data

Array of **DIDEVICEOBJECTDATA** types containing the data to send to the device.

The **lOfs** field of each **DIDEVICEOBJECTDATA** type must contain the instance identifier (not the data offset) for the device object to which the data is directed. (See **DirectInputDeviceObjectInstance.GetType**.) The **lTimeStamp** and **lSequence** members must be 0.

flags

Flags controlling the manner in which data is sent. This may be 0 or the following value:

DISDD_CONTINUE

The device data sent will be overlaid on the previously sent device data. See Remarks.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST

DIERR_NOTACQUIRED

DIERR_REPORTFULL

DIERR_UNPLUGGED

Remarks

There is no guarantee that the individual data elements will be sent in a particular order. However, data sent by successive calls to **SendDeviceData** will not be interleaved. Furthermore, if multiple pieces of data are sent to the same object with a single call, it is unspecified which piece of data is sent.

Consider, for example, a device that can be sent data in packets, each packet describing two pieces of information, call them A and B. Suppose the application attempts to send three data elements: B = 2, A = 1, and B = 0.

The actual device will be sent a single packet. The A field of the packet will contain the value 1, and the B field of the packet will be either 2 or 0.

If the data must to be sent to the device exactly as specified, then three calls to **SendDeviceData** should be performed, each call sending one data element.

In response to the first call, the device will be sent a packet where the A field is blank and the B field contains the value 2.

In response to the second call, the device will be sent a packet where the A field contains the value 1, and the B field is blank.

Finally, in response to the third call, the device will be sent a packet where the A field is blank and the B field contains the value 0.

If the DISDD_CONTINUE flag is set, then the device data sent will be overlaid on the previously sent device data. Otherwise, the device data sent will start from scratch.

For example, suppose a device supports two button outputs, Button0 and Button1. If an application first calls **SendDeviceData** passing "Button0 pressed", then a packet of the form "Button0 pressed, Button1 not pressed" is sent to the device. If the application then makes another call, passing "Button1 pressed" and the DISDD_CONTINUE flag, then a packet of the form "Button0 pressed, Button1 pressed" is sent to the device. However, if the application had not passed the DISDD_CONTINUE flag, the packet sent to the device would have been "Button0 not pressed, Button1 pressed".

DirectInputDevice8.SendForceFeedbackCommand

#Sends a command to the device's force-feedback system.

object.**SendForceFeedbackCommand(***flags* **As Long)**

Parts

object
Resolves to a **DirectInputDevice8** object.

flags
A single value indicating the desired change in state. The value may be one of the members of the **CONST_DISFFCFLAGS** enumeration.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INPUTLOST
DIERR_NOTACQUIRED
DIERR_REPORTFULL
DIERR_UNPLUGGED

IDH_DirectInputDevice8.SendForceFeedbackCommand_dinput_vb

DirectInputDevice8.SetActionMap

#Sets the data format for a device and maps application-defined actions to device objects. It also sets the buffer size for buffered data.

Note

The device must be unacquired prior to calling this method.

object.**SetActionMap(***format* **As DIACTIONFORMAT, _**
userName **As String, _**
flags **As Long)**

Parts

object
Resolves to a **DirectInputDevice8** object.

format
DIACTIONFORMAT type that specifies information about the action map to be applied and about the buffer size for buffered device data.

userName
Value of type **String** that specifies the name of the user.

flags
Value of type **Long** that specifies how the action map is applied. Can be one of the constants of the **CONST_DIDSAMFLAGS** enumeration.

Error Codes

If the method fails, **Err.Number** can be set to one of the following values.

DIERR_ACQUIRED
DIERR_INVALIDPARAM

Remarks

This method provides the mechanism to change action-to-control mapping from the device defaults. An application must use this method to map its actions to virtual controls.

The user name passed to this method binds a set of action mappings for a device to a specific user. Settings are automatically saved to disk when they differ from the currently applied map. Applications that accept input from multiple users should be very careful when applying action maps to the system mouse or keyboard, as the action maps for each user may conflict.

The method can be called only when the device is not acquired.

IDH_DirectInputDevice8.SetActionMap_dinput_vb

See Also

DirectInputDevice8.BuildActionMap

DirectInputDevice8.SetCommonDataFormat

#Sets the input data format for standard devices.

object.**SetCommonDataFormat(** _
format **As CONST_DICOMMONDATAFORMATS)**

Parts

object
Resolves to a **DirectInputDevice8** object.

format
One of the **CONST_DICOMMONDATAFORMATS** enumeration, identifying the data format to use for the device.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_ACQUIRED
DIERR_INVALIDPARAM

Remarks

The data format must be set before the device can be acquired by using the **DirectInputDevice8.Acquire** method. It is necessary to set the data format only once. The data format cannot be changed while the device is acquired.

See Also

DirectInputDevice8.SetDataFormat

DirectInputDevice8.SetCooperativeLevel

#Establishes the cooperative level for this instance of the device. The cooperative level determines how this instance of the device interacts with other instances of the device and the rest of the system.

object.**SetCooperativeLevel(***hwnd* **As Long**, _

IDH_DirectInputDevice8.SetCommonDataFormat_dinput_vb
IDH_DirectInputDevice8.SetCooperativeLevel_dinput_vb

flags **As CONST_DISCLFLAGS)**

Parts

object
Resolves to a **DirectInputDevice8** object.

hwnd
Window handle to be associated with the device. This parameter must be a valid top-level window handle that belongs to the process. The window associated with the device must not be destroyed while it is still active in a Microsoft® DirectInput® device.

flags
Flags that describe the cooperative level associated with the device. The flags are constants of the **CONST_DISCLFLAGS** enumeration.

The following combinations of flags are valid.

Flags	Meaning	Valid for
DISCL_NONEXCLUSIVE **Or** DISCL_BACKGROUND	Others can acquire device in exclusive or nonexclusive mode; your application has access to data at all times.	All.
DISCL_NONEXCLUSIVE **Or** DISCL_FOREGROUND	Others can acquire device in exclusive or nonexclusive mode; your application has access to data only when in the foreground.	All.
DISCL_EXCLUSIVE **Or** DISCL_BACKGROUND	Others can acquire device in nonexclusive mode; your application has access to data at all times.	Joystick.
DISCL_EXCLUSIVE **Or** DISC_FOREGROUND	Others can acquire device in nonexclusive mode; your application has access to data only when in the foreground.	All. Valid for mouse but prevents Microsoft® Windows® from displaying the cursor.

For keyboards, DISCL_NOWINKEY can be also be included with DISCL_NONEXCLUSIVE to disable the Windows logo key. the Windows logo key is always disabled when DISCL_EXCLUSIVE is specified. Note, however, that DISCL_NOWINKEY has no effect when the default action-mapping UI is displayed, and the Windows logo key operates normally as long as that UI is present.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM
DIERR_INVALIDHANDLE

Remarks

No two applications (or instances of the same application) can have a device acquired in exclusive mode at the same time. This is primarily a security feature; it prevents input intended for one application from going to another that may be running concurrently.

If the system mouse is acquired in exclusive mode, then the pointer will be removed from the screen until the device is unacquired.

Applications must call this method before acquiring the device by using the **DirectInputDevice8.Acquire** method.

See Also

Cooperative Levels

DirectInputDevice8.SetDataFormat

#Sets the data format for a Microsoft® DirectInput® device that is not a standard mouse, keyboard, or joystick.

object.**SetDataFormat(***format* **As DIDATAFORMAT, _**
formatArray() **As DIOBJECTDATAFORMAT))**

Parts

object
Resolves to a **DirectInputDevice8** object.

format
A **DIDATAFORMAT** type that describes the format of the data the device should return.

formatArray
Array of **DIOBJECTDATAFORMAT** types describing data formats for objects on the device.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_ACQUIRED
DIERR_INVALIDPARAM

IDH_DirectInputDevice8.SetDataFormat_dinput_vb

Remarks

The data format must be set before the device can be acquired by using the **DirectInputDevice8.Acquire** method. It is necessary to set the data format only once. The data format cannot be changed while the device is acquired.

See Also

DirectInputDevice8.SetActionMap, **DirectInputDevice8.SetCommonDataFormat**

DirectInputDevice8.SetEventNotification

#Sets the event notification status. This method specifies an event that is to be set when the device state changes. It is also used to turn off event notification.

object.**SetEventNotification(***hEvent* **As Long)**

Parts

object
Resolves to a **DirectInputDevice8** object.

hEvent
Handle to the event that is to be set when the device state changes, or 0 to disable notification.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_ACQUIRED
DIERR_HANDLEEXISTS
DIERR_INVALIDPARAM

Remarks

A device state change is defined as any of the following:

- A change in the position of an axis
- A change in the state (pressed or released) of a button
- A change in the direction of a POV control
- Loss of acquisition

IDH_DirectInputDevice8.SetEventNotification_dinput_vb

You must call this method with the *hEvent* parameter set to 0 before destroying the event.

The event notification handle cannot be changed while the device is acquired.

See Also

Polling and Event Notification, **DirectXEvent8**

DirectInputDevice8.SetProperty

#Sets properties that define the device behavior.

object.**SetProperty(** _
guid **As String**, _
propertyInfo **As Any)**

Parts

object
Resolves to a **DirectInputDevice8** object.

guid
Identifier of the property to be set. The following property values are predefined for an input device and can be passed as strings:

DIPROP_AUTOCENTER
Specifies whether device objects are self-centering. This setting applies to the entire device, rather than to any particular object, so the **lHow** member of the associated **DIPROPLONG** type must be DIPH_DEVICE.

The **lData** member can be one of the following values from the **CONST_DINPUT** enumeration.

DIPROPAUTOCENTER_OFF: The device should not automatically center when the user releases the device. An application that uses force feedback should disable autocentering before playing effects.

DIPROPAUTOCENTER_ON: The device should automatically center when the user releases the device.

Not all devices support the autocenter property.

DIPROP_AXISMODE
Sets the axis mode. The value being set (either DIPROPAXISMODE_ABS or DIPROPAXISMODE_REL from the **CONST_DINPUT** enumeration) must be specified in the **lData** member of the associated **DIPROPLONG** type.

This setting applies to the entire device, so the **lHow** member of the **DIPROPLONG** type must be set to DIPH_DEVICE.

DIPROP_BUFFERSIZE
Sets the input-buffer size. See Remarks.

IDH_DirectInputDevice8.SetProperty_dinput_vb

This setting applies to the entire device, so the **lHow** member of the associated **DIPROPLONG** type must be set to DIPH_DEVICE.

DIPROP_CALIBRATIONMODE

Enables the application to specify whether Microsoft® DirectInput® should retrieve calibrated or uncalibrated data from an axis. By default, DirectInput retrieves calibrated data.

Setting the calibration mode for the entire device is equivalent to setting it for each axis individually.

The **lData** member of the **DIPROPLONG** type may be one of the following values:

DIPROPCALIBRATIONMODE_COOKED: DirectInput should return data after applying calibration information. This is the default mode.

DIPROPCALIBRATIONMODE_RAW: DirectInput should return raw, uncalibrated data. This mode is typically used only by Control Panel–type applications.

Note that setting a device into raw mode causes the dead zone, saturation, and range settings to be ignored.

DIPROP_DEADZONE

Sets the value for the dead zone of a joystick, in the range 0 to 10,000, where 0 indicates there is no dead zone, 5,000 indicates that the dead zone extends over 50 percent of the physical range of the axis on both sides of center, and 10,000 indicates that the entire physical range of the axis is dead. When the axis is within the dead zone, it is reported as being at the center of its range.

This setting can be applied to either the entire device or to a specific axis.

DIPROP_RANGE

Sets the range of values an object can possibly report. The minimum and maximum values are taken from the **lMin** and **lMax** members of the associated **DIPROPRANGE** type.

For some devices, this is a read-only property.

You cannot set a reverse range; **lMax** must be greater than **lMin**.

DIPROP_SATURATION

Sets the value for the saturation zones of a joystick, in the range 0 to 10,000. The saturation level is the point at which the axis is considered to be at its most extreme position. For example, if the saturation level is set to 9,500, then the axis reaches the extreme of its range when it has moved 95 percent of the physical distance from its center position (or from the dead zone).

This setting can be applied to either the entire device or to a specific axis.

propertyInfo

A **DIPROPLONG** type containing data for properties that take a single value, or a **DIPROPRANGE** type containing data for properties that take a pair of values.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM
DIERR_OBJECTNOTFOUND
DIERR_UNSUPPORTED

Remarks

The buffer size determines the amount of data that the buffer can hold between calls to the **DirectInputDevice8.GetDeviceData** method before data is lost. This value may be set to 0 to indicate that the application will not be reading buffered data from the device. If the buffer size in the **lData** member of the **DIPROPLONG** type is too large to be supported by the device, the largest possible buffer size is set. To determine whether the requested buffer size was set, retrieve the buffer-size property and compare the result with the value you previously attempted to set.

See Also

DirectInputDevice8.GetProperty

DirectInputDevice8.Unacquire

#Releases access to the device.

object.**Unacquire()**

Parts

object
Resolves to a **DirectInputDevice8** object.

Error Codes

None.

See Also

DirectInputDevice8.Acquire

DirectInputDevice8.WriteEffectToFile

#Writes a force-feedback effect to a file.

IDH_DirectInputDevice8.Unacquire_dinput_vb
IDH_DirectInputDevice8.WriteEffectToFile_dinput_vb

object.**WriteEffectToFile(_**
filename **As String, _**
flags **As Long, _**
guid **As String, _**
name **As String, _**
CoverEffect **As DIEFFECT)**

Parts

object
Resolves to a **DirectInputDevice8** object.

filename
Name of the RIFF file. If the file exists, it is overwritten.

flags
Flags which control how the effect should be written. Can be one of the following values from the **CONST_DIFEFFLAGS** enumeration.

DIFEF_DEFAULT
Do not write the effect if it is not a standard type.

DIFEF_INCLUDENONSTANDARD
Write the effect even if it is not a standard type.

guid
Identifier of the effect type, such as "GUID_RampForce". For a list of standard effects, see **DirectInputDevice8.CreateEffect**. For nonstandard effects, this parameter must be a string representation of the numerical GUID.

name
Friendly name of the effect.

CoverEffect
DIEFFECT type that specifies parameters of the effect.

Error Codes

If the method fails, an error is raised and **Err.Number** may be one of the following error codes.

DIERR_INVALIDPARAM

Remarks

This method is chiefly of interest for force-authoring applications.

See Also

DirectInputDevice8.CreateEffectFromFile

DirectInputDeviceInstance8

#The **DirectInputDeviceInstance8** class is used to obtain information about an instance of a Microsoft® DirectInput® device.

An object of this class is returned by the **DirectInputDevice8.GetDeviceInfo** and **DirectInputEnumDevices8.GetItem** method.

The **DirectInputDeviceInstance8** class has the following methods:

Information	**GetDevType**
	GetGuidFFDriver
	GetGuidInstance
	GetGuidProduct
	GetInstanceName
	GetProductName
	GetUsage
	GetUsagePage

DirectInputDeviceInstance8.GetDevType

#Retrieves the device type and subtype.

object.**GetDevType() As Long**

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns a device type specifier. This value is a combination of a type (in the least significant byte) and subtype (in the next most significant byte), optionally combined using **Or** with DIDEVTYPE_HID, which specifies a Human Interface Device (HID). The following constants are from the **CONST_DI8DEVICETYPE** and **CONST_DI8DEVICESUBTYPE** enumerations.

DI8DEVTYPE_1STPERSON
First-person action game device. The following subtypes are defined.

DI8DEVTYPE1STPERSON_LIMITED
Device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPE1STPERSON_SHOOTER

IDH_DirectInputDeviceInstance8_dinput_vb
IDH_DirectInputDeviceInstance8.GetDevType_dinput_vb

Device designed for first-person shooter games.

DI8DEVTYPE1STPERSON_SIXDOF
Device with six degrees of freedom; that is, three lateral axes and three rotational axes.

DI8DEVTYPE1STPERSON_UNKNOWN
Unknown subtype.

DI8DEVTYPE_DEVICE
Device that does not fall into another category.

DI8DEVTYPE_DEVICECTRL
Input device used to control another type of device from within the context of the application. The following subtypes are defined.

DI8DEVTYPEDEVICECTRL_COMMSSELECTION
Control used to make communications selections.

DI8DEVTYPEDEVICECTRL_HARDWIRED
Device which must use its default configuration and cannot be remapped.

DI8DEVTYPEDEVICECTRL_UNKNOWN
Unknown subtype.

DI8DEVTYPE_DRIVING
Device for steering. The following subtypes are defined.

DI8DEVTYPEDRIVING_COMBINEDPEDALS
Steering device that reports acceleration and brake pedal values from a single axis.

DI8DEVTYPEDRIVING_DUALPEDALS
Steering device that reports acceleration and brake pedal values from separate axes.

DI8DEVTYPEDRIVING_HANDHELD
Hand-held steering device.

DI8DEVTYPEDRIVING_THREEPEDALS
Steering device that reports acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPE_FLIGHT
Controller for flight simulation. The following subtypes are defined.

DI8DEVTYPEFLIGHT_LIMITED
Flight controller that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEFLIGHT_RC
Flight device based on a remote control for model aircraft.

DI8DEVTYPEFLIGHT_STICK
Joystick.

DI8DEVTYPEFLIGHT_YOKE
Yoke.

DI8DEVTYPE_GAMEPAD
Gamepad. The following subtypes are defined.

DI8DEVTYPEGAMEPAD_LIMITED
Gamepad that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_STANDARD
Standard gamepad that provides the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_TILT
Gamepad that can report x-axis and y-axis data based on the attitude of the controller.

DI8DEVTYPE_JOYSTICK
Joystick. The following subtypes are defined.

DI8DEVTYPEJOYSTICK_LIMITED
Joystick that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_STANDARD
Standard gamepad that provides the minimum number of device objects for action mapping.

DI8DEVTYPEJOYSTICK_STANDARD
Standard joystick that provides the minimum number of device objects for action mapping.

DI8DEVTYPE_KEYBOARD
Keyboard or keyboard-like device. The following subtypes are defined.

DI8DEVTYPEKEYBOARD_UNKNOWN
Subtype could not be determined.

DI8DEVTYPEKEYBOARD_PCXT
IBM PC/XT 83-key keyboard.

DI8DEVTYPEKEYBOARD_OLIVETTI
Olivetti 102-key keyboard.

DI8DEVTYPEKEYBOARD_PCAT
IBM PC/AT 84-key keyboard.

DI8DEVTYPEKEYBOARD_PCENH
IBM PC Enhanced 101/102-key or Microsoft® Natural® keyboard.

DI8DEVTYPEKEYBOARD_NOKIA1050
Nokia 1050 keyboard.

DI8DEVTYPEKEYBOARD_NOKIA9140
Nokia 9140 keyboard.

DI8DEVTYPEKEYBOARD_NEC98
Japanese NEC PC98 keyboard.

DI8DEVTYPEKEYBOARD_NEC98LAPTOP

Japanese NEC PC98 laptop keyboard.

DI8DEVTYPEKEYBOARD_NEC98106
Japanese NEC PC98 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPAN106
Japanese 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPANAX
Japanese AX keyboard.

DI8DEVTYPEKEYBOARD_J3100
Japanese J3100 keyboard.

DI8DEVTYPE_MOUSE
A mouse or mouse-like device (such as a trackball). The following subtypes are defined.

DI8DEVTYPEMOUSE_ABSOLUTE
Mouse that returns absolute axis data.

DI8DEVTYPEMOUSE_FINGERSTICK
Fingerstick.

DI8DEVTYPEMOUSE_TOUCHPAD
Touchpad.

DI8DEVTYPEMOUSE_TRACKBALL
Trackball.

DI8DEVTYPEMOUSE_TRADITIONAL
Traditional mouse.

DI8DEVTYPEMOUSE_UNKNOWN
Subtype could not be determined.

DI8DEVTYPE_REMOTE
Remote-control device. The only defined subtype is DI8DEVTYPEREMOTE_UNKNOWN.

DI8DEVTYPE_SCREENPOINTER
Screen pointer. The following subtypes are defined.

DI8DEVTYPESCREENPTR_UNKNOWN
Unknown subtype.

DI8DEVTYPESCREENPTR_LIGHTGUN
Light gun.

DI8DEVTYPESCREENPTR_LIGHTPEN
Light pen.

DI8DEVTYPESCREENPTR_TOUCH
Touch screen.

DI8DEVTYPE_SUPPLEMENTAL
Specialized device with functionality unsuitable for the main control of an application, such as pedals used with a wheel. The following subtypes are defined.

DI8DEVTYPESUPPLEMENTAL_2NDHANDCONTROLLER
Secondary handheld controller.

DI8DEVTYPESUPPLEMENTAL_COMBINEDPEDALS
Device whose primary function is to report acceleration and brake pedal values from a single axis.

DI8DEVTYPESUPPLEMENTAL_DUALPEDALS
Device whose primary function is to report acceleration and brake pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_HANDTRACKER
Device that tracks hand movement.

DI8DEVTYPESUPPLEMENTAL_HEADTRACKER
Device that tracks head movement.

DI8DEVTYPESUPPLEMENTAL_RUDDERPEDALS
Device with rudder pedals.

DI8DEVTYPESUPPLEMENTAL_SHIFTER
Device that reports gear selection from an axis.

DI8DEVTYPESUPPLEMENTAL_SHIFTSTICKGATE
Device that reports gear selection from button states.

DI8DEVTYPESUPPLEMENTAL_SPLITTHROTTLE
Device whose primary function is to report at least two throttle values. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_THREEPEDALS
Device whose primary function is to report acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_THROTTLE
Device whose primary function is to report a single throttle value. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_UNKNOWN
Unknown subtype.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

To look for a particular subtype of device such as a wheel, check both the type and the subtype.

DirectInputDeviceInstance8.GetGuidFFDriver

#Retrieves the unique identifier for the force-feedback driver.

object.**GetGuidFFDriver() As String**

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the GUID for the force feedback driver, in string form. This identifier is established by the manufacturer of the driver.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

Using GUIDs

DirectInputDeviceInstance8.GetGuidInstance

#Retrieves the unique identifier for the instance of the device.

object.**GetGuidInstance() As String**

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the GUID for the device instance, in string form.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

IDH_DirectInputDeviceInstance8.GetGuidFFDriver_dinput_vb
IDH_DirectInputDeviceInstance8.GetGuidInstance_dinput_vb

Remarks

An application can save the instance GUID into a configuration file and use it at a later time. Instance GUIDs are specific to a particular computer. An instance GUID obtained from one computer is unrelated to instance GUIDs on another.

See Also

DirectInputDeviceInstance8.GetGuidProduct, Using GUIDs

DirectInputDeviceInstance8.GetGuidProduct

#Retrieves the manufacturer's unique identifier for the device.

*object.***GetGuidProduct() As String**

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the GUID for the product, in string form. This identifier is established by the manufacturer of the device.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceInstance8.GetGuidInstance, Using GUIDs

DirectInputDeviceInstance8.GetInstanceName

#Retrieves the name of the device instance.

*object***.GetInstanceName() As String**

Parts

object

IDH_DirectInputDeviceInstance8.GetGuidProduct_dinput_vb
IDH_DirectInputDeviceInstance8.GetInstanceName_dinput_vb

Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the friendly name for the instance—for example, "Joystick 1".

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceInstance8.GetProductName

DirectInputDeviceInstance8.GetProductName

#Retrieves the product name of the device.

object.**GetProductName() As String**

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the friendly name for the product—for example, "Microsoft® SideWinder®".

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceInstance8.GetInstanceName

DirectInputDeviceInstance8.GetUsage

#Retrieves the usage code for Human Interface Devices (HIDs).

object.**GetUsage() As Integer**

IDH_DirectInputDeviceInstance8.GetProductName_dinput_vb
IDH_DirectInputDeviceInstance8.GetUsage_dinput_vb

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the HID usage code for HID devices, or 0.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceInstance8.GetUsagePage

DirectInputDeviceInstance8.GetUsagePage

#Retrieves the usage page for Human Interface Devices (HIDs).

object.**GetUsagePage() As Integer**

Parts

object
Resolves to a **DirectInputDeviceInstance8** object.

Return Values

Returns the usage page or 0 for non-HIDs.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceInstance8.GetUsage

DirectInputDeviceObjectInstance

#The **DirectInputDeviceObjectInstance** class represents an object on a Microsoft® DirectInput® device, such as a button or axis.

IDH_DirectInputDeviceInstance8.GetUsagePage_dinput_vb
IDH_DirectInputDeviceObjectInstance_dinput_vb

A **DirectInputDeviceObjectInstance** object is returned by the **DirectInputDevice8.GetObjectInfo** and **DirectInputEnumDeviceObjects.GetItem** methods.

This class has the following methods:

Information	**GetCollectionNumber**
	GetDesignatorIndex
	GetDimension
	GetExponent
	GetFlags
	GetGuidType
	GetName
	GetOfs
	GetType
	GetUsage
	GetUsagePage

DirectInputDeviceObjectInstance.GetCollectionNumber

#Retrieves the number of the HID link collection to which the device object belongs.

object.**GetCollectionNumber() As Integer**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the number of the collection or 0 if the object is not part of an HID.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDeviceObjectInstance.GetDesignatorIndex

#Retrieves the designator index for an object on a Human Interface Device (HID).

IDH_DirectInputDeviceObjectInstance.GetCollectionNumber_dinput_vb
IDH_DirectInputDeviceObjectInstance.GetDesignatorIndex_dinput_vb

object.**GetDesignatorIndex() As Integer**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns an index that refers to a designator in the HID physical descriptor.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDeviceObjectInstance.GetDimension

#Retrieves a Human Interface Device (HID)code for the dimensional units in which the object's value is reported.

object.**GetDimension() As Long**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns a code for the dimensional units in which the object's value is reported, if known, or 0 if not known.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceObjectInstance.GetExponent

DirectInputDeviceObjectInstance.GetExponent

#Retrieves the exponent to associate with the dimensional units of the device object.

IDH_DirectInputDeviceObjectInstance.GetDimension_dinput_vb
IDH_DirectInputDeviceObjectInstance.GetExponent_dinput_vb

object.**GetExponent() As Integer**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the exponent to associate with the dimension, if known. Dimensional units are always integral, so an exponent may be needed to convert them to non-integral types.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceObjectInstance.GetDimension

DirectInputDeviceObjectInstance.GetFlags

#Retrieves the flags associated with the device object.

object.**GetFlags() As CONST_DIDEVICEOBJINSTANCEFLAGS**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns one or more members of the **CONST_DIDEVICEOBJINSTANCEFLAGS** enumeration.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDeviceObjectInstance.GetGuidType

#Retrieves the unique identifier of the object type.

IDH_DirectInputDeviceObjectInstance.GetFlags_dinput_vb
IDH_DirectInputDeviceObjectInstance.GetGuidType_dinput_vb

object.**GetGuidType() As String**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Can return one of the following string identifiers, representing the unique identifier for the object type. If the object type has a GUID not represented in the following list, a string representing the actual GUID is returned. If the object type does not have a GUID, an empty string is returned.

GUID_XAxis
The horizontal axis. For example, it may represent the left-right motion of a mouse.

GUID_YAxis
The vertical axis. For example, it may represent the forward-backward motion of a mouse.

GUID_ZAxis
The z-axis. For example, it may represent rotation of the wheel on a mouse, or movement of a throttle control on a joystick.

GUID_RxAxis
Rotation around the x-axis.

GUID_RyAxis
Rotation around the y-axis.

GUID_RzAxis
Rotation around the z-axis (often a rudder control).

GUID_Slider
A slider axis.

GUID_Button
A button on a mouse.

GUID_Key
A key on a keyboard.

GUID_POV
A point-of-view indicator or "hat".

GUID_Unknown
Unknown.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDeviceObjectInstance.GetName

#Retrieves the friendly name of the device object.

object.**GetName() As String**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the name of the object—for example, "X-Axis" or "Right Shift."

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputDeviceObjectInstance.GetOfs

#Retrieves the offset of the device object's data within the data format for the device.

object.**GetOfs() As Long**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the offset within the data format at which data is reported for this object. This value can be used to identify the object in method calls and types that accept the DIPH_BYOFFSET flag.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDevice8.GetObjectInfo, **DIPROPLONG**, **DIPROPRANGE**

IDH_DirectInputDeviceObjectInstance.GetName_dinput_vb
IDH_DirectInputDeviceObjectInstance.GetOfs_dinput_vb

DirectInputDeviceObjectInstance.GetType

#Retrieves the type and instance identifier of the object.

object.**GetType() As Long**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the device type that describes the object. This value is a combination of **CONST_DIDFTFLAGS** flags that describe the object type (axis, button, and so forth) and contains the object instance number in the middle 16 bits.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

To extract the object instance ID, use the following operation:

```
Dim ObjID as Long
ObjID = (diObj.GetType And &HFFFF00) \ 256
```

DirectInputDeviceObjectInstance.GetUsage

#Retrieves the Human Interface Device (HID) usage code for the device object.

object.**GetUsage() As Integer**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the HID usage associated with the object, if known. HIDs always report a usage. Non-HID devices might report a usage; if they do not, the return value is zero.

IDH_DirectInputDeviceObjectInstance.GetType_dinput_vb
IDH_DirectInputDeviceObjectInstance.GetUsage_dinput_vb

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceObjectInstance.GetUsagePage

DirectInputDeviceObjectInstance.GetUsagePage

#Retrieves the Human Interface Device (HID) usage page for the device object.

object.**GetUsagePage() As Integer**

Parts

object
Resolves to a **DirectInputDeviceObjectInstance** object.

Return Values

Returns the HID usage page associated with the object, if known. (HIDs) will always report a usage page. Non-HID devices might report a usage page; if they do not, the return value is zero.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

DirectInputDeviceObjectInstance.GetUsage

DirectInputEffect

#An object of the **DirectInputEffect** class represents a force-feedback effect created by an application. The object is created by using the **DirectInputDevice8.CreateEffect** method.

This class has the following methods:

Control	**Download**
	SetParameters
	Start

IDH_DirectInputDeviceObjectInstance.GetUsagePage_dinput_vb
IDH_DirectInputEffect_dinput_vb

	Stop
	Unload
Information	**GetEffectGuid**
	GetEffectStatus
	GetParameters

DirectInputEffect.Download

#Places the effect on the device. If the effect is already on the device, then the existing effect is updated to match the values set by the **DirectInputEffect.SetParameters** method.

object.**Download()**

Parts

object
Resolves to a **DirectInputEffect** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_NOTINITIALIZED
DIERR_DEVICEFULL
DIERR_INCOMPLETEEFFECT
DIERR_INPUTLOST
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_INVALIDPARAM
DIERR_EFFECTPLAYING

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

It is valid to update an effect while it is playing. The semantics of such an operation are explained in the reference for **DirectInputEffect.SetParameters**.

DirectInputEffect.GetEffectGuid

#Retrieves the GUID or GUID alias for the effect represented by the **DirectInputEffect** object.

object.**GetEffectGuid() As String**

IDH_DirectInputEffect.Download_dinput_vb
IDH_DirectInputEffect.GetEffectGuid_dinput_vb

Parts

object
Resolves to a **DirectInputEffect** object.

Return Values

Returns the GUID or alias that was passed to **DirectInputDevice8.CreateEffect**, such as "GUID_ConstantForce".

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

See Also

Using GUIDs

DirectInputEffect.GetEffectStatus

#Retrieves the status of an effect.

object.**GetEffectStatus() As Long**

Parts

object
Resolves to a **DirectInputEffect** object.

Return Values

Returns status flags for the effect. The value may be zero, or one or more of the following constants of the **CONST_DIEGESFLAGS** enumeration.

DIEGES_PLAYING
The effect is playing.

DIEGES_EMULATED
The effect is emulated.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

DirectInputEffect.GetParameters

#Retrieves information about an effect.

IDH_DirectInputEffect.GetEffectStatus_dinput_vb
IDH_DirectInputEffect.GetParameters_dinput_vb

object.**GetParameters(***effectinfo* **As DIEFFECT)**

Parts

object
Resolves to a **DirectInputEffect** object.

effectinfo
DIEFFECT type to receive the effect parameters.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_INVALIDPARAM
DIERR_MOREDATA
DIERR_NOTINITIALIZED

DirectInputEffect.SetParameters

#Sets the characteristics of an effect.

object.**SetParameters(***effectinfo* **As DIEFFECT, _**
flags **As CONST_DIEPFLAGS)**

Parts

object
Resolves to a **DirectInputEffect** object.

effectinfo
DIEFFECT type containing effect parameters.

flags
Flags from the **CONST_DIEPFLAGS** enumeration specifying which portions of the effect information are to be set and how the downloading of the parameters should be handled.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_NOTINITIALIZED
DIERR_INCOMPLETEEFFECT
DIERR_INPUTLOST
DIERR_INVALIDPARAM

IDH_DirectInputEffect.SetParameters_dinput_vb

DIERR_EFFECTPLAYING

Remarks

To determine which parameters can be dynamically updated while the effect is playing, use the **DirectInputEnumEffects.GetDynamicParams** method.

The **DirectInputEffect.SetParameters** method automatically downloads the effect, but this behavior can be suppressed by setting the DIEP_NODOWNLOAD flag. If automatic download has been suppressed, then you can manually download the effect by calling **DirectInputEffect.Download**.

If the effect is playing while the parameters are changed, then the new parameters take effect as if they were the parameters when the effect started.

For example, suppose a periodic effect with a duration of three seconds is started. After two seconds, the direction of the effect is changed. The effect will then continue for one additional second in the new direction. The envelope, phase, amplitude, and other parameters of the effect continue smoothly as if the direction had not changed.

In the same scenario, if after two seconds the duration of the effect were changed to 1.5 seconds, then the effect would stop.

Normally, if the driver cannot update the parameters of a playing effect, the driver is permitted to stop the effect, update the parameters, and then restart the effect. Passing the DIEP_NORESTART flag suppresses this behavior. If the driver cannot update the parameters of an effect while it is playing, the error code DIERR_EFFECTPLAYING is returned and the parameters are not updated.

No more than one of the DIEP_NODOWNLOAD, DIEP_START, and DIEP_NORESTART flags should be set. (It is also valid to pass none of them.)

These three flags control download and playback behavior as follows:

If DIEP_NODOWNLOAD is set, the effect parameters are updated but not downloaded to the device.

If the DIEP_START flag is set, the effect parameters are updated and downloaded to the device, and the effect is started just as if the **DirectInputEffect.Start** method had been called with the *iterations* parameter set to 1 and with no flags. (Combining the update with DIEP_START is slightly faster than calling **Start** separately, because it requires less information to be transmitted to the device.)

If neither DIEP_NODOWNLOAD nor DIEP_START is set and the effect is not playing, then the parameters are updated and downloaded to the device.

If neither DIEP_NODOWNLOAD nor DIEP_START is set and the effect is playing, then the parameters are updated if the device supports on-the-fly updating. Otherwise the behavior depends on the state of the DIEP_NORESTART flag. If it is set, the error code DIERR_EFFECTPLAYING is returned. If it is clear, the effect is stopped, the parameters are updated, and the effect is restarted.

DirectInputEffect.Start

#Begins playing an effect. If the effect is already playing, it is restarted from the beginning. If the effect has not been downloaded or has been modified since its last download, then it will be downloaded before being started. This default behavior can be suppressed by passing the DIES_NODOWNLOAD flag.

object.**Start(***iterations* **As Long,** *flags* **As Long)**

Parts

object
Resolves to a **DirectInputEffect** object.

iterations
Number of times to play the effect in sequence. The envelope is re-articulated with each iteration.

To play the effect exactly once, pass 1. To play the effect repeatedly until explicitly stopped, pass -1. To play the effect until explicitly stopped without re-articulating the envelope, modify the effect parameters with the **DirectInputEffect.SetParameters** method and change the **lDuration** member of **DIEFFECT** to -1.

flags
Flags from the **CONST_DIESFLAGS** enumeration that describe how the effect should be played by the device. The value can be zero or one or more flags.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_INVALIDPARAM
DIERR_INCOMPLETEEFFECT
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED
DIERR_UNSUPPORTED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

Not all devices support multiple iterations.

IDH_DirectInputEffect.Start_dinput_vb

DirectInputEffect.Stop

#Stops an effect that is playing.

object.**Stop()**

Parts

object
Resolves to a **DirectInputEffect** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

DirectInputEffect.Unload

#Removes the effect from the device. If the effect is playing, it is automatically stopped before it is unloaded.

object.**Unload()**

Parts

object
Resolves to a **DirectInputEffect** object.

Error Codes

If the method fails, an error is raised and **Err.Number** may be set to one of the following:

DIERR_INPUTLOST
DIERR_INVALIDPARAM
DIERR_NOTEXCLUSIVEACQUIRED
DIERR_NOTINITIALIZED

IDH_DirectInputEffect.Stop_dinput_vb
IDH_DirectInputEffect.Unload_dinput_vb

Remarks

The device must be acquired at the exclusive cooperative level for this method to succeed.

DirectInputEnumDeviceObjects

#The **DirectInputEnumDeviceObjects** class enumerates the Microsoft® DirectInputDevice8 objects installed on a system. This object is created and filled with data as a result of a call to **DirectInputDevice8.GetDeviceObjectsEnum** method.

This class has the following methods:

Information	**GetCount**
	GetItem

DirectInputEnumDeviceObjects.GetCount

#Returns the number of items in the **DirectInputEnumDeviceObjects** collection.

object.**GetCount() As Long**

Parts

object
Resolves to a **DirectInputEnumDeviceObjects** object.

Return Values

Returns the number of device objects enumerated for the device.

Error Codes

None.

DirectInputEnumDeviceObjects.GetItem

#Retrieves an object describing the specified device object.

object.**GetItem(***index* **As Long) _**
As DirectInputDeviceObjectInstance

IDH_DirectInputEnumDeviceObjects_dinput_vb
IDH_DirectInputEnumDeviceObjects.GetCount_dinput_vb
IDH_DirectInputEnumDeviceObjects.GetItem_dinput_vb

Parts

object
Resolves to a **DirectInputEnumDeviceObjects** object.

index
Index of the enumerated item to retrieve.

Return Values

Returns a **DirectInputDeviceObjectInstance** object whose methods can be used to retrieve information about the device object.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

To get the number of entries in the **DirectInputEnumDeviceObjects** first call the **DirectInputEnumDeviceObjects.GetCount** method.

Applications should not rely on this method to determine whether certain keyboard keys or indicator lights are present, as these objects might be enumerated even though they are not present. Although the basic set of available objects can be determined from the device subtype, there is no reliable way of determining whether extra objects such as the menu key are available.

DirectInputEnumDevices8

#The **DirectInputEnumDevices8** class enumerates the Microsoft® DirectInput® devices installed on a system. This object is created and filled with data as a result of a call to the **DirectInput8.GetDIDevices** and **DirectInput8.GetDevicesBySemantics** methods.

This class has the following methods:

GetCount

GetItem

DirectInputEnumDevices8.GetCount

#Returns the number of Microsoft® DirectInput® devices in the **DirectInputEnumDevices8** collection.

object.**GetCount() As Long**

IDH_DirectInputEnumDevices8_dinput_vb
IDH_DirectInputEnumDevices8.GetCount_dinput_vb

Parts

object
Resolves to a **DirectInputEnumDevices8** object.

Return Values

Returns the number of DirectInput® devices in the **DirectInputEnumDevices8** collection.

Error Codes

None.

DirectInputEnumDevices8.GetItem

#Returns information about an enumerated device.

object.**GetItem(***index* **As Long) As DirectInputDeviceInstance8**

Parts

object
Resolves to a **DirectInputEnumDevices8** object.

index
Index of the device in the **DirectInputEnumDevices8** collection.

Return Values

Returns a **DirectInputDeviceInstance8** object whose methods can be used to retrieve information about the device.

Error Codes

If the method fails, an error is raised and **Err.Number** is set.

Remarks

To get the number of entries in **DirectInputEnumDevices8** first call the **DirectInputEnumDevices8.GetCount** method.

IDH_DirectInputEnumDevices8.GetItem_dinput_vb

DirectInputEnumEffects

#An object of the **DirectInputEnumEffects** class represents a collection of supported force-feedback effects on a device, and is used for retrieving information about the effects. The object is created by using the **DirectInputDevice8.GetEffectsEnum** method.

The class has the following methods:

Information	**GetCount**
	GetDynamicParams
	GetEffectGuid
	GetName
	GetStaticParams
	GetType

DirectInputEnumEffects.GetCount

#Returns the number of enumerated effects in the collection.

object.**GetCount() As Long**

Parts

object
Resolves to a **DirectInputEnumEffects** object.

Return Values

Returns the number of items in the collection.

Error Codes

If the method fails, it raises an error and **Err.Number** is set.

DirectInputEnumEffects.GetDynamicParams

#Retrieves information about which effect parameters can be changed without stopping the effect, using the **DirectInputEffect.SetParameters** method.

object.**GetDynamicParams(***i* **As Long) As CONST_DIEPFLAGS**

IDH_DirectInputEnumEffects_dinput_vb
IDH_DirectInputEnumEffects.GetCount_dinput_vb
IDH_DirectInputEnumEffects.GetDynamicParams_dinput_vb

Parts

object
Resolves to a **DirectInputEnumEffects** object.

i
Index (1-based) of the effect in the collection.

Return Values

Returns flags from the **CONST_DIEPFLAGS** enumeration specifying which parameters can be dynamically updated while the effect is playing.

Error Codes

If the method fails, it raises an error and **Err.Number** is set.

See Also

DirectInputEffect.SetParameters

DirectInputEnumEffects.GetEffectGuid

#Retrieves the GUID for the supported effect. This GUID can be passed to the **DirectInputDevice8.CreateEffect** method.

object.**GetEffectGuid(***i* **As Long) As String**

Parts

object
Resolves to a **DirectInputEnumEffects** object.

i
Index (1-based) of the effect in the collection.

Return Values

Returns the GUID in string form. For standard effects, the return value is an alias such as "GUID_ConstantForce".

Error Codes

If the method fails, it raises an error and **Err.Number** is set.

See Also

Using GUIDs

IDH_DirectInputEnumEffects.GetEffectGuid_dinput_vb

DirectInputEnumEffects.GetName

#Retrieves the name of the effect.

object.**GetName(***i* **As Long) As String**

Parts

object
Resolves to a **DirectInputEnumEffects** object.

i
Index (1-based) of the effect in the collection.

Return Values

Returns the name of the effect, such as "Constant Force".

Error Codes

If the method fails, it raises an error and **Err.Number** is set.

DirectInputEnumEffects.GetStaticParams

#Retrieves information about effect parameters supported on the device.

object.**GetStaticParams(***i* **As Long) As CONST_DIEPFLAGS**

Parts

object
Resolves to a **DirectInputEnumEffects** object.

i
Index (1-based) of the effect in the collection.

Return Values

Returns flags from the **CONST_DIEPFLAGS** enumeration specifying which parameters are supported by the effect.

Error Codes

If the method fails, it raises an error and **Err.Number** is set.

IDH_DirectInputEnumEffects.GetName_dinput_vb
IDH_DirectInputEnumEffects.GetStaticParams_dinput_vb

DirectInputEnumEffects.GetType

#Retrieves information about the type and capabilities of the effect.

object.**GetType(***i* **As Long) As CONST_DIEFTFLAGS**

Parts

object
Resolves to a **DirectInputEnumEffects** object.

i
Index (1-based) of the effect in the collection.

Return Values

Returns flags from the **CONST_DIEFTFLAGS** enumeration giving information about the effect**.**

Error Codes

If the method fails, it raises an error and **Err.Number** is set.

Remarks

The effect type is stored in the low byte and can be retrieved by a bitwise **And** with &HFF.

Types

This section contains information on the following types used with Microsoft® DirectInput®.

- **DIACTION**
- **DIACTIONFORMAT**
- **DICOLORSET**
- **DICONDITION**
- **DICONFIGUREDEVICESPARAMS**
- **DICONSTANTFORCE**
- **DIDATAFORMAT**
- **DIDEVCAPS**
- **DIDEVICEIMAGEINFO**
- **DIDEVICEIMAGEINFOHEADER**

IDH_DirectInputEnumEffects.GetType_dinput_vb

- **DIDEVICEOBJECTDATA**
- **DIEFFECT**
- **DIENVELOPE**
- **DIJOYSTATE**
- **DIJOYSTATE2**
- **DIKEYBOARDSTATE**
- **DIMOUSESTATE**
- **DIMOUSESTATE2**
- **DIOBJECTDATAFORMAT**
- **DIPERIODICFORCE**
- **DIPROPGUIDANDPATH**
- **DIPROPLONG**
- **DIPROPRANGE**
- **DIPROPSTRING**
- **DIRAMPFORCE**

DIACTION

#Specifies and receives information about the mapping of a single application action to a device object or virtual control.

```
Type DIACTION
    ActionName As String
    lAppData As Long
    lFlags As Long
    lHow As Long
    lObjId As Long
    lSemantic As Long
End Type
```

Members

ActionName
Application-defined name of the action. This string is displayed by the device property sheet when **DirectInput8.ConfigureDevices** is called. If **sResIDString** is not an empty string, this member is ignored.

lAppData
Value to be returned to the application when the state of the control associated with the action changes. This value is returned in the **lUserData** member of the **DIDEVICEOBJECTDATA** type retrieved by **DirectInputDevice8.GetDeviceData**.

IDH_DIACTION_dinput_vb

lFlags
Flags used to request specific attributes or processing. Can be zero or one or more of the values in the **CONST_DIAFLAGS** enumeration.

lHow
Mapping mechanism used by Microsoft® DirectInput® to configure the action. This member receives one of the members of the **CONST_DIAHFLAGS** enumeration when the type is returned by **DirectInputDevice8.BuildActionMap**.

This member is ignored when the action map is passed to **DirectInputDevice8.SetActionMap**.

lObjId
Control identifier.

lSemantic
For a joystick, a predefined action mapping constant for this application genre, representing a virtual control. See **CONST_DIGENRE**.

This value can also be one of the **CONST_DIMAPFLAGS** enumeration to map the action to a virtual control not defined in a genre.

For a keyboard, mouse, or Microsoft DirectPlay® voice device, this value is a constant that represents a particular device object.

See Also

CONST_DIKEYBOARDFLAGS, **CONST_DIMOUSEFLAGS**, **CONST_DIVOICEFLAGS**

DIACTIONFORMAT

#Specifies and receives information about an action map.

```
Type DIACTIONFORMAT
    ActionArray() As DIACTION
    ActionMapName As String
    guidActionMap As String
    lActionCount As Long
    lAxisMax As Long
    lAxisMin As Long
    lBufferSize As Long
    lDataSize As Long
    lGenre As Long
End Type
```

IDH_DIACTIONFORMAT_dinput_vb

Members

ActionArray
Array of **DIACTION** types, each of which describes how an action maps to a virtual control or device object, and how the mapping information should be displayed to the user.

ActionMapName
Friendly name for the action map.

guidActionMap
GUID that identifies the action map. This value can be used by device manufacturers to tune mappings for a specific title.

lActionCount
Number of elements in **ActionArray**.

lAxisMax
Maximum value for the range of scaled data to be returned for all axes. This value is ignored for a specific action axis if the DIA_NORANGE flag is set in the **DIACTION** type for that action. See Remarks.

lAxisMin
Minimum value for the range of scaled data to be returned for all axes. This value is ignored for a specific action axis if the DIA_NORANGE flag is set in the **DIACTION** type for that action. See Remarks.

lBufferSize
Value that specifies the number of input data packets in the buffer for each device to which this action map is applied. See Remarks.

lDataSize
Value that specifies the size of immediate data to be returned by the device, in bytes. When this type is passed to **DirectInputDevice8.SetActionMap**, this value should be four times the number of elements in the array of **DIACTION** types passed to the method.

lGenre
Genre of the application. One of the members of the CONST_DIGENRE enumeration whose names begin with DIVIRTUAL.

Remarks

The **lAxisMax** and **lAxisMin** members are valid only for axis actions and should be set to zero for all other actions. Setting them before passing this type to **DirectInputDevice8.SetActionMap** is equivalent to setting the range properties by passing a **DIPROPRANGE** type to **DirectInputDevice8.SetProperty**.

The buffer size must be set to a value greater than zero if you want to retrieve data by using **DirectInputDevice8.GetDeviceData**. Applications that set the **lBufferSize** member to the desired buffer size before passing this type to **DirectInputDevice8.SetActionMap** do not need to set the DIPROP_BUFFERSIZE property by using **DirectInputDevice8.SetProperty**.

DICOLORSET

#Describes a set of colors used to draw the device configuration user interface. It is part of the **DICONFIGUREDEVICESPARAMS** TYPE.

```
Type DICOLORSET
   cAreaFill As Long
   cBorder As Long
   cCalloutHighlight As Long
   cCalloutLine As Long
   cControlFill As Long
   cHighlightFill As Long
   cTextFore As Long
   cTextHighlight As Long
End Type
```

Members

cAreaFill
Fill color for areas outside controls such as tabs and buttons.

cBorder
Border color, used to display lines around controls such as tabs and buttons.

cCalloutHighlight
Color used to display callout lines.

cCalloutLine
Color used to display callout lines.

cControlFill
Fill color for controls such as tabs and buttons.

cHighlightFill
Fill color for highlighted controls.

cTextFore
Foreground text color.

cTextHighlight
Foreground color for highlighted text.

Remarks

Text background color is always transparent.

IDH_DICOLORSET_dinput_vb

DICONDITION

#Describes parameters for a force-feedback condition in the **DIEFFECT** type.

```
Type DICONDITION
   lDeadBand As Long
   lNegativeCoefficient As Long
   lNegativeSaturation As Long
   lOffset As Long
   lPositiveCoefficient As Long
   lPositiveSaturation As Long
End Type
```

Members

lDeadBand

The region around **lOffset** where the condition is not active, in the range 0 to 10,000. In other words, the condition is not active between **lOffset** - **lDeadBand** and **lOffset** + **lDeadBand**.

lNegativeCoefficient

The coefficient constant on the negative side of the offset, in the range -10,000 to +10,000.

If the device does not support separate positive and negative coefficients, then the value of **lNegativeCoefficient** is ignored and the value of **lPositiveCoefficient** is used as both the positive and negative coefficients.

lNegativeSaturation

The maximum force output on the negative side of the offset, in the range 0 to 10,000.

If the device does not support force saturations, then the value of this member is ignored.

If the device does not support separate positive and negative saturations, then the value of **lNegativeSaturation** is ignored and the value of **lPositiveSaturation** is used as both the positive and negative saturations.

lOffset

The offset for the condition, in the range -10,000 to +10,000.

lPositiveCoefficient

The coefficient constant on the positive side of the offset, in the range -10,000 to +10,000.

lPositiveSaturation

The maximum force output on the positive side of the offset, in the range 0 to 10,000.

If the device does not support force saturation, then the value of this member is ignored.

IDH_DICONDITION_dinput_vb

Remarks

Different types of conditions will interpret the parameters differently, but the basic idea is that force resulting from a condition is equal to *A(q - q0)* where *A* is a scaling coefficient, *q* is some metric, and *q0* is the neutral value for that metric.

The preceding simplified formula must be adjusted if a nonzero dead band is provided. If the metric is less than **lOffset - lDeadBand**, then the resulting force is given by the following formula:

force = **lNegativeCoefficient** * (*q* - (**lOffset - lDeadBand**))

Similarly, if the metric is greater than **lOffset** + **lDeadBand**, then the resulting force is given by the following formula:

force = **lPositiveCoefficient** * (*q* - (**lOffset** + **lDeadBand**))

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.

DICONFIGUREDEVICESPARAMS

#Contains information for the device configuration property sheet. It is used by the **DirectInput8.ConfigureDevices** method.

```
Type DICONFIGUREDEVICESPARAMS
    ActionFormats() As DIACTIONFORMAT
    DDSTarget As Unknown
    dics As DICOLORSET
    FormatCount As Long
    hwnd As Long
    UserCount As Long
    UserNames() As String
End Type
```

Members

ActionFormats

Array of **DIACTIONFORMAT** types that contains action-mapping information for each genre used by the game. On input, this array contains action-to-control mappings and strings to display as callouts for each mapping. The configuration interface displays the genres in its drop-down list in the same order as in the array.

DDSTarget

IDH_DICONFIGUREDEVICESPARAMS_dinput_vb

Direct3DSurface8 object for the configuration user interface. The device image is alpha-blended over the background surface onto the target surface.

Applications that are not using DirectDraw and applications that are using DirectDraw but are windowed rather than using the full screen may pass NULL in this parameter. This will cause Microsoft DirectInput® to use the Windows Graphics Device Interface (GDI) functions to draw the configuration user interface image.

dics

DICOLORSET type that describes the color scheme to apply to the configuration user interface.

FormatCount

Number of elements in the **ActionFormats** array.

hwnd

Handle to the top-level window of the calling application. The member is needed only for applications that run in windowed mode, and is otherwise ignored.

UserCount

Number of elements in the **UserNames** array. Zero is an invalid value. If the UserCount value exceeds the number of entries actually in the array at UserNames(), the method fails, returning DIERR_INVALIDPARAM. If the UserNames array is set to NULL, the value in UserCount is ignored.

UserNames

Array of user names. This parameter can be set to NULL to request default names, the number of which is determined by **UserCount**. If the application passes more than **UserCount** names, only the names within the count are used. If an application specifies names that are different from the names currently assigned to devices, ownership is revoked for all devices, a default name is created for the mismatched name, and the interface shows "(No User)" for all devices.

DICONSTANTFORCE

[#]Describes parameters for a constant force in the **DIEFFECT** type.

```
Type DICONSTANTFORCE
   lMagnitude As Long
End Type
```

Members

lMagnitude

Magnitude of the effect, in the range -10,000 to +10,000. If an envelope is applied to this effect, then the value represents the magnitude of the sustain. If no envelope is applied, then the value represents the amplitude of the entire effect.

IDH_DICONSTANTFORCE_dinput_vb

DIDATAFORMAT

#Describes a device's data format. This type is used with the **DirectInputDevice8.SetDataFormat** method.

```
Type DIDATAFORMAT
   dataSize As Long
   lFlags As CONST_DIDATAFORMATFLAGS
   lObjSize As Long
   numObjs As Long
End Type
```

Members

dataSize
Size of a data packet returned by the device, in bytes. This value must be a multiple of 4 and must exceed the largest offset value for an object's data within the data packet.

lFlags
Flags describing other attributes of the data format. This value can be one of the **CONST_DIDATAFORMATFLAGS** enumeration.

lObjSize
Size of the **DIOBJECTDATAFORMAT** type, in bytes.

numObjs
Number of objects for which data is to be returned.

Remarks

Applications need to create a **DIDATAFORMAT** type only for nonstandard devices. For the mouse, keyboard, and joystick, you set the data format by using **DirectInputDevice8.SetCommonDataFormat**.

DIDEVCAPS

#Describes capabilities of a Microsoft® DirectInput® device. This type is used with the **DirectInputDevice8.GetCapabilities** method.

```
Type DIDEVCAPS
   lAxes As Long
   lButtons As Long
   lDevType As Long
   lDriverVersion As Long
   lFFMinTimeResolution As Long
   lFFSamplePeriod As Long
```

IDH_DIDATAFORMAT_dinput_vb
IDH_DIDEVCAPS_dinput_vb

```
    lFirmwareRevision As Long
    lFlags As CONST_DIDEVCAPSFLAGS
    lHardwareRevision As Long
    lPOVs As Long
End Type
```

Members

lAxes
Number of axes available on the device.

lButtons
Number of buttons available on the device.

lDevType
A packed value containing information about the type and subtype of the device. The value is identical to that returned by the **DirectInputDeviceInstance8.GetDevType** method.

lDriverVersion
The version number of the device driver.

lFFMinTimeResolution
The minimum amount of time, in microseconds, that the device can resolve when playing force-feedback effects. The device rounds any times to the nearest supported increment. For example, if the value of **lFFMinTimeResolution** is 1000, then the device would round any times to the nearest millisecond.

lFFSamplePeriod
The minimum time between playback of consecutive raw force commands.

lFirmwareRevision
Specifies the firmware revision of the device.

lFlags
Flags associated with the device. This value can be a combination of the constants of the **CONST_DIDEVCAPSFLAGS** enumeration.

lHardwareRevision
The hardware revision of the device.

lPOVs
Number of point-of-view controllers available on the device.

Remarks

The semantics of version numbers are left to the manufacturer of the device. The only guarantee is that newer versions will have larger numbers.

DIDEVICEIMAGEINFO

#Carries information required to display a device image, or an overlay image with a callout. This type is passed to the **DirectInputDevice8.GetImageInfo** method as an array within a **DIDEVICEIMAGEINFOHEADER** type.

```
Type DIDEVICEIMAGEINFO
    CalloutLine(0 To 4) As POINT
    CalloutRect As RECT
    flags As Long
    ImagePath As String
    ObjId As Long
    ViewID As Long
    OverlayRect As RECT
    TextAlign As Long
    ValidPts As Long
End Type
```

Members

CalloutLine

Array of types that specify coordinates of the points describing a callout line. A callout line connects a device control to a caption for the game action. Each line can have one to four segments. This member is valid only if the DIDIFT_OVERLAY flag is present in **flags**.

CalloutRect

Type that specifies coordinates of the rectangle in which the game action string is displayed. The application handles clipping. This member is valid only if the DIDIFT_OVERLAY flag is present in **flags**.

flags

Flag that describes the intended use of the image. This member can be one of the following values from the **CONST_DIDIFTFLAGS** enumeration.

DIDIFT_CONFIGURATION

The file is used to display the current configuration of actions on the device. Overlay image coordinates are relative to the upper left corner of the configuration image.

DIDIFT_OVERLAY

The file (if provided) is an overlay for a particular control on the configuration image. The **ViewID**, **OverlayRect**, **ObjID**, **CalloutLine**, **CalloutRect**, and **TextAlign** members are valid and contain data used to display the overlay and callout information for a single control on the device. If no file is provided (null path string), all other pertinent members are relevant except **rdOverlay**.

ImagePath

Fully qualified path to the file that contains an image of the device.

ObjId

IDH_DIDEVICEIMAGEINFO_dinput_vb

Control identifier, combining object type flags and an instance value, to which an overlay image corresponds for this device. For more information on this identifier, see **DirectInputDeviceObjectInstance.GetType**. This member is valid only if the DIDIFT_OVERLAY flag is present in **flags**.

OverlayOffset

Offset of the device configuration image over which this overlay is to be displayed. This member is valid only if the DIDIFT_OVERLAY flag is present in **flags**.

OverlayRect

Rectangle, using coordinates relative to the top-left pixel of the device configuration image, in which the overlay image should be painted. This member is valid only if the DIDIFT_OVERLAY flag is present in **flags**.

TextAlign

Value that specifies the alignment of the text in the rectangle described by the **CalloutRect** member. Must be one horizontal alignment flag combined with one vertical alignment flag from the **CONST_DIDALFLAGS** enumeration.

The following horizontal alignment flags are defined.

DIDAL_LEFTALIGNED
Text is aligned on the left border.

DIDAL_CENTERED
Text is horizontally centered.

DIDAL_RIGHTALIGNED
Text is aligned on the right border.

The following vertical alignment flags are defined.

DIDAL_MIDDLE
The text is vertically centered.

DIDAL_TOPALIGNED
The text is aligned on the top border.

DIDAL_BOTTOMALIGNED
The text is aligned on the bottom border.

This member is valid only if the DIDIFT_OVERLAY flag is present in **flags**.

ValidPts

Number of points in the **CalloutLine** array.

ViewID

For device view images (DIDIFT_CONFIGURATION), this is the ID of the device view. For device control overlays (DIDIFT_OVERLAY), this value refers to the device view (by ID) over which an image and callout information should be displayed.

DIDEVICEIMAGEINFOHEADER

#Contains information about device images. This type is passed to the **DirectInputDevice8.GetImageInfo** method. On return, it contains information about the device and the images that represent it in the device configuration property sheet.

```
Type DIDEVICEIMAGEINFOHEADER
    Axes As Long
    Buttons As Long
    ImageCount As Long
    Images() As DIDEVICEIMAGEINFO
    POVs As Long
    Views As Long
End Type
```

Members

Axes
Number of axes on the device.

Buttons
Number of buttons on the device.

ImageCount
Number of elements in the **Images** array.

Images
Array of **DIDEVICEIMAGEINFO** types describing device images and views, overlay images, and callout-string coordinates.

POVs
Number of point-of-view controllers on the device.

Views
Number of views of this device.

Remarks

Before passing this type to **DirectInputDevice8.GetImageInfo**, the application must initialize the **ImageCount** and **Images** members as follows:

```
' Assume that didev is a DirectInputDevice8 object

Dim count as Integer
Dim diImageInfoHdr

count = didev.GetImageInfoCount
With diImageInfoHdr
  ImageCount = count
```

IDH_DIDEVICEIMAGEINFOHEADER_dinput_vb

```
  ReDim Images(count)
End With
```

Members of the type other than **ImageCount** are filled with data by the method call.

See Also

DirectInputDevice8.GetImageInfoCount

DIDEVICEOBJECTDATA

#Contains raw buffered device information. This type is used with the **DirectInputDevice8.GetDeviceData** method.

```
Type DIDEVICEOBJECTDATA
   lData As Long
   lOfs As Long
   lSequence As Long
   lTimeStamp As Long
   lUserData As Long
End Type
```

Members

lData

Data obtained from the device.

For axis input, if the device is in relative axis mode, then the relative axis motion is reported. If the device is in absolute axis mode, then the absolute axis coordinate is reported.

For button input, only the low byte of **lData** is significant. The high bit of the low byte is set if the button went down; it is clear if the button went up.

lOfs

Offset into the current data format of the object whose data is being reported—that is, the location where the data would have been stored if it had been obtained by a call to the **DirectInputDevice8.GetDeviceStateX** method where **X** refers to the specific device, for instance **GetDeviceStateMouse**. If the device is accessed as a keyboard, you can determine which key generated the event by comparing this value with the members of the **CONST_DIKEYFLAGS** enumeration. For the mouse and joystick, the value in **lOfs** is equivalent to the byte offset of the button or axis within the **DIMOUSESTATE**, **DIJOYSTATE**, or **DIJOYSTATE2** type (depending on the data format that was established by using **DirectInputDevice8.SetCommonDataFormat**.) Constants for these offsets are contained in the **CONST_DIMOUSEOFS** and **CONST_DIJOYSTICKOFS** enumerations. If a custom data format has been set

IDH_DIDEVICEOBJECTDATA_dinput_vb

by using **DirectInputDevice8.SetDataFormat**, then **lOfs** is the offset of the device object's place in the custom data format.

lSequence

Microsoft® DirectInput® sequence number for this event. All input events are assigned an increasing sequence number. This allows events from different devices to be sorted chronologically. Because this value can wrap around, care must be taken when comparing two sequence numbers.

lTimeStamp

System time at which the input event was generated, in milliseconds. This value wraps around approximately every 50 days.

lUserData

Application-defined action value assigned to this object in the last call to **DirectInputDevice8.SetActionMap**. This is the value in the **lAppData** member of the **DIACTION** type associated with the object. Ignore this value if you are not using action mapping.

When the typestructure is used with the **SendDeviceData** method, this member must be zero.

DIEFFECT

#Describes a force-feedback effect. It is passed to the **DirectInputDevice8.CreateEffect** and **DirectInputEffect.SetParameters** methods in order to set parameters for an effect. Existing parameters are retrieved in this type through the **DirectInputEffect.GetParameters** method.

```
Type DIEFFECT
    bUseEnvelope As Long
    conditionFlags As CONST_DICONDITIONFLAGS
    conditionX As DICONDITION
    conditionY As DICONDITION
    constantForce As DICONSTANTFORCE
    envelope As DIENVELOPE
    lDuration As Long
    lFlags As Long
    lGain As Long
    lSamplePeriod As Long
    lStartDelay As Long
    lTriggerButton As Long
    lTriggerRepeatInterval As Long
    periodicForce As DIPERIODICFORCE
    rampForce As DIRAMPFORCE
    x As Long
    y As Long
End Type
```

IDH_DIEFFECT_dinput_vb

Members

bUseEnvelope
True if the envelope described in the **envelope** member is to be applied to the effect.

conditionFlags
Flags from the **CONST_DICONDITIONFLAGS** enumeration that determine use of the **conditionX** and **conditionY** members.

conditionX
DICONDITION type describing parameters of the condition on the x-axis. Ignored for other types of effects.

conditionY
DICONDITION type describing parameters of the condition on the y-axis. Ignored for other types of effects.

constantForce
DICONSTANTFORCE type describing parameters of a constant force. Ignored for other types of effects.

envelope
DIENVELOPE type describing parameters of an envelope to be applied to the effect. Valid only if the **bUseEnvelope** member is True.

lDuration
Duration of the effect, in microseconds. A value of -1 indicates infinite duration. If an envelope is applied to an effect of infinite duration, then the attack will be applied, followed by an infinite sustain.

lFlags
Zero or more members of the **CONST_DIEFFFLAGS** enumeration specifying how other members are to be interpreted.

lGain
The gain to be applied to the effect, in the range 0 to 10,000. The gain is a scaling factor applied to all magnitudes of the effect and its envelope.

lSamplePeriod
The period, in microseconds, at which the device samples the effect—in other words, the granularity of changes in force. A value of 0 indicates that the default playback sample rate should be used.

If the device is not capable of sampling the effect at the specified rate, it will choose the supported rate that is closest to the requested value.

Setting a custom sample period can be used for special effects. For example, playing a sine wave with an artificially large sample period results in a rougher texture.

lStartDelay
Time, in microseconds, the device should wait after a **DirectInputEffect.Start** call before playing the effect. If this value is 0, then effect playback begins immediately.

lTriggerButton

Offset value of the button that will trigger the effect. This should be one of the members of the **CONST_DIJOYSTICKOFS** enumeration, or DIEP_NOTRIGGER to indicate that the effect does not have a trigger button.

lTriggerRepeatInterval

The interval, in microseconds, between the end of one playback and the start of the next when the effect is triggered by a button press and the button is held down. Setting this value to -1 suppresses repetition.

Not all devices support trigger repeat.

periodicForce

DIPERIODICFORCE type describing parameters of a periodic effect. Ignored for other types of effects.

rampForce

DIRAMPFORCE type describing parameters of a ramp force. Ignored for other types of effects.

x

Direction of the effect. Normally, this is the amount by which the direction is rotated from north (usually the negative y-axis), in hundredths of degrees. Thus a value of 0 indicates a force pushing toward the user, a value of 9000 indicates a force pushing from the user's right, and so on. In this case, **y** should be 0.

If **lFlags** contains DIEFF_CARTESIAN, this is a Cartesian value describing the relative amount of force on the x-axis. For example, if **x** = -1 and **y** = 1, the direction of the force is from the southwest. For more information, see Effect Direction.

y

If **lFlags** contains DIEFF_CARTESIAN, this is a Cartesian value describing the relative amount of force on the y-axis. Otherwise it should be 0.

DIENVELOPE

#Describes parameters for an envelope in the **DIEFFECT** type.

```
Type DIENVELOPE
    lAttackLevel As Long
    lAttackTime As Long
    lFadeLevel As Long
    lFadeTime As Long
End Type
```

Members

lAttackLevel

Amplitude for the start of the envelope, relative to the baseline (offset), in the range 0 to 10,000. If the effect's type-specific data does not specify a baseline, then the amplitude is relative to zero.

IDH_DIENVELOPE_dinput_vb

lAttackTime
The time, in microseconds, to reach the sustain level.

lFadeLevel
Amplitude for the end of the envelope, relative to the baseline, in the range 0 to 10,000. If the effect's type-specific data does not specify a baseline, then the amplitude is relative to zero.

lFadeTime
The time, in microseconds, to reach the fade level.

DIJOYSTATE

#Describes the state of a joystick device. (This term includes other controllers such as game pads and steering wheels). This type is used with the **DirectInputDevice8.GetDeviceStateJoystick** method.

```
Type DIJOYSTATE
    Buttons(0 To 31) As Byte
    POV(0 To 3) As Long
    rx As Long
    ry As Long
    rz As Long
    slider(0 To 1) As Long
    x As Long
    y As Long
    z As Long
End Type
```

Members

Buttons
Array of buttons. The high-order bit of the byte is set if the corresponding button is down and clear if the button is up or does not exist.

POV
Direction controllers (such as point-of-view hats). The position is indicated in hundredths of degrees clockwise from north (away from the user). The center position is normally reported as -1; but see Remarks. For indicators that have only five positions, the value for a controller will be -1, 0, 9,000, 18,000, or 27,000.

rx
X-axis rotation. If the joystick does not have this, the value is 0.

ry
Y-axis rotation. If the joystick does not have this axis, the value is 0.

rz

IDH_DIJOYSTATE_dinput_vb

Z-axis rotation. If the joystick does not have this axis, the value is 0.

slider
Two additional axes whose semantics depend on the joystick. Use the **DirectInputDevice8.GetObjectInfo** method to obtain semantic information about these values.

x
X-axis, usually the left-right movement of a stick.

y
Y-axis, usually the forward-backward movement of a stick.

z
Z-axis, often the throttle control. If the joystick does not have this axis, the value is zero.

Remarks

You must prepare the device for joystick-style access by calling the **DirectInputDevice8.SetCommonDataFormat** method, passing the DIFORMAT_JOYSTICK format constant.

If an axis is in relative mode, then the appropriate member contains the change in position. If it is in absolute mode, then the member contains the absolute axis position.

Some drivers report the centered position of the POV indicator as 65,535. You can determine whether an indicator is centered as follows:

```
Dim POVCentered as Boolean
POVCentered = MyDijoystate.POV(0) And &HFFFF
```

Note

Under Microsoft® DirectX® 7, sliders on some joysticks could be assigned to the Z axis, with subsequent code retrieving data from that member. Using DirectX 8, those same sliders will be assigned to the **slider** array. This should be taken into account when porting applications to DirectX 8. Make any necessary alterations to ensure that slider data is retrieved from the **slider** array.

See Also

DIJOYSTATE2

DIJOYSTATE2

#Describes the state of a joystick device with extended capabilities. This type is used with the **DirectInputDevice8.GetDeviceStateJoystick2** method.

IDH_DIJOYSTATE2_dinput_vb

Most applications do not need to use this type, which is for highly specialized controllers including force-feedback devices. For standard game controllers, use the **DIJOYSTATE** type and obtain data by calling **DirectInputDevice8.GetDeviceStateJoystick**.

```
Type DIJOYSTATE2
    arx As Long
    ary As Long
    arz As Long
    aslider (0 to 1) As Long
    ax As Long
    ay As Long
    az As Long
    Buttons(0 To 127) As Byte
    frx As Long
    fry As Long
    frz As Long
    fslider(0 To 1) As Long
    fx As Long
    fy As Long
    fz As Long
    POV(0 To 3) As Long
    rx As Long
    ry As Long
    rz As Long
    slider(0 To 1) As Long
    vrx As Long
    vry As Long
    vrz As Long
    vslider(0 To 1) As Long
    vx As Long
    vy As Long
    vz As Long
    x As Long
    y As Long
    z As Long
End Type
```

Members

arx
X-axis angular acceleration.

ary
Y-axis angular acceleration.

arz
Z-axis angular acceleration.

aslider
Extra axis accelerations.

ax
X-axis acceleration.

ay
Y-axis acceleration.

az
Z-axis acceleration.

Buttons
Array of buttons. The high-order bit of the byte is set if the corresponding button is down and clear if the button is up or does not exist.

frx
X-axis torque.

fry
Y-axis torque.

frz
Z-axis torque.

fslider
Extra axis forces.

fx
X-axis force.

fy
Y-axis force.

fz
Z-axis force.

POV
Array of direction controllers, such as point-of-view hats. The position is indicated in hundredths of degrees clockwise from north (away from the user). The center position is normally reported as -1; but see Remarks. For indicators that have only five positions, the value for a controller will be -1, 0, 9,000, 18,000, or 27,000.

rx
X-axis rotation. If the joystick does not have this, the value is 0.

ry
Y-axis rotation. If the joystick does not have this axis, the value is 0.

rz
Z-axis rotation (often called the rudder). If the joystick does not have this axis, the value is zero.

slider
Two additional axes whose semantics depend on the joystick. Use the **DirectInputDevice8.GetObjectInfo** method to obtain semantic information about these values.

vrx

X-axis angular velocity.

vry
Y-axis angular velocity.

vrz
Z-axis angular velocity.

vslider
Array of extra axis velocities.

vx
X-axis velocity.

vy
Y-axis velocity.

vz
Z-axis velocity.

x
Information about the joystick x-axis (usually the left-right movement of a stick).

y
Information about the joystick y-axis (usually the forward-backward movement of a stick).

z
Information about the joystick z-axis (often the throttle control). If the joystick does not have this axis, the value is zero.

Remarks

You must prepare the device for access to a joystick with extended capabilities by calling the **DirectInputDevice8.SetCommonDataFormat** method, passing the DIFORMAT_JOYSTICK2 data format variable.

If an axis is in relative mode, then the appropriate member contains the change in position. If it is in absolute mode, then the member contains the absolute axis position.

Some drivers report the centered position of the POV indicator as 65,535. You can determine whether an indicator is centered as follows:

```
Dim POVCentered as Boolean
POVCentered = MyDijoystate2.POV(0) And &HFFFF
```

Note

Under Microsoft® DirectX® 7, sliders on some joysticks could be assigned to the Z axis, with subsequent code retrieving data from that member. Using DirectX 8, those same sliders will be assigned to the **slider** array. This should be taken into account when porting applications to DirectX 8. Make any necessary alterations to ensure that slider data is retrieved from the **slider** array.

DIKEYBOARDSTATE

#Describes the state of keyboard keys. This type is used with the **DirectInputDevice8.GetDeviceStateKeyboard** method.

```
Type DIKEYBOARDSTATE
   key(0 To 255) As Byte
End Type
```

Members

key

Array of key states. The array can be indexed by using members of the **CONST_DIKEYFLAGS** enumeration. For each key, the high bit is set if the key is down, and clear if the key is up or does not exist.

Remarks

The following example checks to see if the ESC key is being pressed:

```
Dim keyState as DIKEYBOARDSTATE
' diDevice is a valid DirectInputDevice8 object.

Call diDevice.GetDeviceStateKeyboard(keyState)
If (keyState.key(DIK_ESCAPE) And &H80) Then
   ' Key is down
End If
```

DIMOUSESTATE

#Describes the state of a mouse device with up to four buttons. This type is used with the **DirectInputDevice8.GetDeviceStateMouse** method.

```
Type DIMOUSESTATE
   Buttons(0 To 3) As Byte
   lX As Long
   lY As Long
   lZ As Long
End Type
```

Members

Buttons

Array of button states. The high-order bit of a byte is set if the corresponding button is down.

lX

X-axis.

IDH_DIKEYBOARDSTATE_dinput_vb
IDH_DIMOUSESTATE_dinput_vb

lY
Y-axis.

lZ
Z-axis, typically a wheel. If the mouse does not have a z-axis, then the value is 0.

Remarks

Immediate data is returned in this type from a device that has been prepared by passing the DIFORMAT_MOUSE constant to the **DirectInputDevice8.SetCommonDataFormat** method.

If an axis is in relative mode, then the appropriate member contains the change in position since the last call to this method. If the axis is in absolute mode, then the member contains the accumulated relative motion in relation to an arbitrary start point. The absolute axis position is not meaningful except in comparison with other absolute axis positions.

See Also

DIMOUSESTATE2

DIMOUSESTATE2

#Describes the state of a mouse device with up to 8 buttons. This type is used with the **DirectInputDevice8.GetDeviceStateMouse2** method.

```
Type DIMOUSESTATE2
    Buttons(0 To 7) As Byte
    lX As Long
    lY As Long
    lZ As Long
End Type
```

Members

Buttons
Array of button states. The high-order bit of a byte is set if the corresponding button is down.

lX
X-axis.

lY
Y-axis.

lZ
Z-axis, typically a wheel. If the mouse does not have a z-axis, the value is 0.

IDH_DIMOUSESTATE2_dinput_vb

Remarks

Immediate data is returned in this type from a device that has been prepared by passing the DIFORMAT_MOUSE2 constant to the **DirectInputDevice8.SetCommonDataFormat** method.

If an axis is in relative mode, then the appropriate member contains the change in position since the last call to this method. If the axis is in absolute mode, then the member contains the accumulated relative motion in relation to an arbitrary start point. The absolute axis position is not meaningful except in comparison with other absolute axis positions.

See Also

DIMOUSESTATE

DIOBJECTDATAFORMAT

#Describes a device object's data format for use with the **DirectInputDevice8.SetDataFormat** method.

```
Type DIOBJECTDATAFORMAT {
   lFlags As CONST_DIDEVICEOBJINSTANCEFLAGS
   lOfs As Long
   lType As CONST_DIDFTFLAGS
   strGuid As String
End Type
```

Members

lFlags

Zero or more of the following values from the **CONST_DIDEVICEOBJINSTANCEFLAGS** enumeration.

DIDOI_ASPECTACCEL
: The object selected by **DirectInputDevice8.SetDataFormat** must report acceleration information.

DIDOI_ASPECTFORCE
: The object selected by **DirectInputDevice8.SetDataFormat** must report force information.

DIDOI_ASPECTPOSITION
: The object selected by **DirectInputDevice8.SetDataFormat** must report position information.

DIDOI_ASPECTVELOCITY
: The object selected by **DirectInputDevice8.SetDataFormat** must report velocity information.

lOfs

IDH_DIOBJECTDATAFORMAT_dinput_vb

Byte offset within the data packet where the data for the input source will be stored. This value must be a multiple of four for **Long** size data, such as axes. It can be byte-aligned for buttons.

lType

Device type that describes the object. It is a combination of values from the **CONST_DIDFTFLAGS** enumeration describing the object type (axis, button, and so forth) and containing the object-instance number in the middle 16 bits.

strGuid

Unique identifier for the type of input source. An empty string indicates that any type of object is permissible.

The following strings can be used in place of actual GUID strings to identify the type of device object:

GUID_XAxis
GUID_YAxis
GUID_ZAxis
GUID_RxAxis
GUID_RyAxis
GUID_RzAxis
GUID_Slider
GUID_Button
GUID_Key
GUID_POV

DIPERIODICFORCE

#Describes parameters for a periodic force in the **DIEFFECT** type.

```
Type DIPERIODICFORCE
   lOffset As Long
   lMagnitude As Long
   lPeriod As Long
   lPhase As Long
End Type
```

Members

lOffset

The offset of the effect. The range of forces generated by the effect will be **lOffset** - **lMagnitude** to **lOffset** + **lMagnitude**. The value of the **lOffset** member is also the baseline for any envelope that is applied to the effect.

lMagnitude

The magnitude of the effect, in the range 0 to 10,000. If an envelope is applied to this effect, then the value represents the magnitude of the sustain. If no envelope is applied, then the value represents the amplitude of the entire effect.

lPeriod

IDH_DIPERIODICFORCE_dinput_vb

The period of the effect, in microseconds.

lPhase

The position in the cycle of the periodic effect at which playback begins, in the range 0 to 35,999. See Remarks.

Remarks

A device driver might not provide support for all values in the **lPhase** member. In this case the value will be rounded off to the nearest supported value.

DIPROPGUIDANDPATH

#Describes properties whose values represent a GUID and a path. Used with the **DirectInputDevice8.GetProperty** method.

```
Type DIPROPGUIDANDPATH
    GUID As String
    lHow As Long
    lObj As Long
    Path As String
End Type
```

Members

GUID

String that receives the GUID.

lHow

Value that specifies how the *lObj* parameter should be interpreted. This value may be one of the members of the **CONST_DIPHFLAGS** enumeration.

If the property pertains to the entire device, **lHow** should be DIPH_DEVICE.

lObj

Object for which the property is to be accessed.

If the **lHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. This value can be retrieved for the object by using the **DirectInputDeviceObjectInstance.GetType** method.

If the **lHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. This value can be obtained by using the **DirectInputDeviceObjectInstance.GetOfs** method.

Path

String that receives the path.

IDH_DIPROPGUIDANDPATH_dinput_vb

DIPROPLONG

#Contains property information that is set on or retrieved from the input device by using the **DirectInputDevice8.SetProperty** and **DirectInputDevice8.GetProperty** methods, where the property is a single value.

```
Type DIPROPLONG
    lData As Long
    lHow As Long
    lObj As Long
End Type
```

Members

lData

The property-specific value being set or retrieved.

lHow

Value specifying how the *lObj* parameter should be interpreted. This value may be one of the members of the **CONST_DIPHFLAGS** enumeration.

If **lObj** is DIPROP_AXISMODE or DIPROP_BUFFERSIZE, **lHow** should be DIPH_DEVICE, because these properties cannot be set for an individual object.

lObj

Object for which the property is to be accessed.

If the **lHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. This value can be retrieved for the object by using the **DirectInputDeviceObjectInstance.GetType** method.

If the **lHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. This value can be obtained by using the **DirectInputDeviceObjectInstance.GetOfs** method.

If **lHow** is DIPH_DEVICE, this value should be 0.

Remarks

All members must be initialized with the proper values before the type is passed to the **SetProperty** method. All members except **lData** must be initialized with the proper values before the type is passed to **GetProperty**.

See Also

DIPROPRANGE

IDH_DIPROPLONG_dinput_vb

DIPROPRANGE

#Contains property information that is set on or retrieved from the input device by using the **DirectInputDevice8.SetProperty** and **DirectInputDevice8.GetProperty** methods, where the property is a range of values.

```
Type DIPROPRANGE
   lHow As Long
   lMax As Long
   lMin As Long
   lObj As Long
End Type
```

Members

lHow

Value specifying how the **lObj** member should be interpreted. This value may be one of the members of the **CONST_DIPHFLAGS** enumeration.

lMax

Upper limit of the range. If the range of the device is unrestricted, this value will be DIPROPRANGE_NOMAX (from the **CONST_DINPUT** enumeration) when the **DirectInputDevice8.GetProperty** method returns.

lMin

Lower limit of the range. If the range of the device is unrestricted, this value will be DIPROPRANGE_NOMIN (from the **CONST_DINPUT** enumeration)when the **DirectInput8Device.GetProperty** method returns.

lObj

Object for which the property is to be accessed.

If the **lHow** member is DIPH_BYID, this member must be the identifier for the object whose property setting is to be set or retrieved. This value can be retrieved for the object by using the **DirectInputDeviceObjectInstance.GetType** method.

If the **lHow** member is DIPH_BYOFFSET, this member must be a data format offset for the object whose property setting is to be set or retrieved. This value can be obtained by using the **DirectInputDeviceObjectInstance.GetOfs** method.

If **lHow** is DIPH_DEVICE, this value should be 0.

See Also

DIPROPLONG

IDH_DIPROPRANGE_dinput_vb

DIPROPSTRING

#Describes string properties. Used with the **DirectInputDevice8.GetProperty** and **DirectInputDevice8.SetProperty** methods.

```
Type DIPROPSTRING
   lHow As Long
   lObj As Long
   PropString As String
End Type
```

Members

lHow
How the **lObj** member should be interpreted.

lObj
Identifier of the object whose property is being retrieved or set.

PropString
String that specifies or receives the property.

DIRAMPFORCE

#Describes parameters for a ramp force in the **DIEFFECT** type.

```
Type DIRAMPFORCE
   lRangeEnd As Long
   lRangeStart As Long
End Type
```

Members

lRangeEnd
The magnitude at the end of the effect, in the range -10,000 to +10,000.

lRangeStart
The magnitude at the start of the effect, in the range -10,000 to +10,000.

Remarks

The duration of a ramp force effect must be finite.

IDH_DIPROPSTRING_dinput_vb
IDH_DIRAMPFORCE_dinput_vb

Enumerations

Microsoft® DirectInput® uses enumerations to group constants in order to take advantage of the statement completion feature of Microsoft Visual Basic®. The enumerations used in DirectInput are as follows:

- **CONST_DI8DEVICESUBTYPE**
- **CONST_DI8DEVICETYPE**
- **CONST_DIAFLAGS**
- **CONST_DIAHFLAGS**
- **CONST_DICDFLAGS**
- **CONST_DICOMMONDATAFORMATS**
- **CONST_DICONDITIONFLAGS**
- **CONST_DIDALFLAGS**
- **CONST_DIDATAFORMATFLAGS**
- **CONST_DIDBAMFLAGS**
- **CONST_DIDEVCAPSFLAGS**
- **CONST_DIDEVICEOBJINSTANCEFLAGS**
- **CONST_DIDFTFLAGS**
- **CONST_DIDGDDFLAGS**
- **CONST_DIDIFTFLAGS**
- **CONST_DIDSAMFLAGS**
- **CONST_DIEDBSFLFLAGS**
- **CONST_DIEFFFLAGS**
- **CONST_DIEFTFLAGS**
- **CONST_DIEGESFLAGS**
- **CONST_DIENUMDEVICESFLAGS**
- **CONST_DIEPFLAGS**
- **CONST_DIESFLAGS**
- **CONST_DIFEFFLAGS**
- **CONST_DIGENRE**
- **CONST_DIGFFSFLAGS**
- **CONST_DIJOYSTICKOFS**
- **CONST_DIKEYBOARDFLAGS**
- **CONST_DIKEYFLAGS**
- **CONST_DIMAPFLAGS**
- **CONST_DIMOUSEFLAGS**
- **CONST_DIMOUSEOFS**

- **CONST_DINPUT**
- **CONST_DINPUTERR**
- **CONST_DIPHFLAGS**
- **CONST_DISCLFLAGS**
- **CONST_DISDDFLAGS**
- **CONST_DISFFCFLAGS**
- **CONST_DIVOICEFLAGS**

CONST_DI8DEVICESUBTYPE

[#]Used to identify device subtypes. A packed value representing the device type and subtype is returned by the **DirectInputDeviceInstance8.GetDevType** method and in the **lDevType** member of the **DIDEVCAPS** type returned by **DirectInputDevice8.GetCapabilities**.

```
Enum CONST_DI8DEVICESUBTYPE
   DI8DEVTYPE_LIMITEDGAMESUBTYPE      =     256 (&H100)
   DI8DEVTYPE1STPERSON_LIMITED        =     256 (&H100)
   DI8DEVTYPE1STPERSON_SHOOTER        =    1024 (&H400)
   DI8DEVTYPE1STPERSON_SIXDOF         =     768 (&H300)
   DI8DEVTYPE1STPERSON_UNKNOWN        =     512 (&H200)
   DI8DEVTYPEDEVICECTRL_COMMSSELECTION =    768 (&H300)
   DI8DEVTYPEDEVICECTRL_HARDWIRED     =    1024 (&H400)
   DI8DEVTYPEDEVICECTRL_UNKNOWN       =     512 (&H200)
   DI8DEVTYPEDRIVING_COMBINEDPEDALS   =     512 (&H200)
   DI8DEVTYPEDRIVING_DUALPEDALS       =     768 (&H300)
   DI8DEVTYPEDRIVING_HANDHELD         =    1280 (&H500)
   DI8DEVTYPEDRIVING_LIMITED          =     256 (&H100)
   DI8DEVTYPEDRIVING_THREEPEDALS      =    1024 (&H400)
   DI8DEVTYPEFLIGHT_LIMITED           =     256 (&H100)
   DI8DEVTYPEFLIGHT_RC                =    1024 (&H400)
   DI8DEVTYPEFLIGHT_STICK             =     512 (&H200)
   DI8DEVTYPEFLIGHT_YOKE              =     768 (&H300)
   DI8DEVTYPEGAMEPAD_LIMITED          =     256 (&H100)
   DI8DEVTYPEGAMEPAD_STANDARD         =     512 (&H200)
   DI8DEVTYPEGAMEPAD_TILT             =     768 (&H300)
   DI8DEVTYPEJOYSTICK_LIMITED         =     256 (&H100)
   DI8DEVTYPEJOYSTICK_STANDARD        =     512 (&H200)
   DI8DEVTYPEKEYBOARD_J3100           =    3072 (&HC00)
   DI8DEVTYPEKEYBOARD_JAPAN106        =    2560 (&HA00)
   DI8DEVTYPEKEYBOARD_JAPANAX         =    2816 (&HB00)
   DI8DEVTYPEKEYBOARD_NEC98           =    1792 (&H700)
   DI8DEVTYPEKEYBOARD_NEC98106        =    2304 (&H900)
```

[#] IDH_CONST_DI8DEVICESUBTYPE_dinput_vb

```
    DI8DEVTYPEKEYBOARD_NEC98LAPTOP     =     2048 (&H800)
    DI8DEVTYPEKEYBOARD_NOKIA1050       =     1280 (&H500)
    DI8DEVTYPEKEYBOARD_NOKIA9140       =     1536 (&H600)
    DI8DEVTYPEKEYBOARD_OLIVETTI        =      512 (&H200)
    DI8DEVTYPEKEYBOARD_PCAT            =      768 (&H300)
    DI8DEVTYPEKEYBOARD_PCENH           =     1024 (&H400)
    DI8DEVTYPEKEYBOARD_PCXT            =      256 (&H100)
    DI8DEVTYPEKEYBOARD_UNKNOWN         =        0 (&H000)
    DI8DEVTYPEMOUSE_ABSOLUTE           =     1536 (&H600)
    DI8DEVTYPEMOUSE_FINGERSTICK        =      768 (&H300)
    DI8DEVTYPEMOUSE_TOUCHPAD           =     1024 (&H400)
    DI8DEVTYPEMOUSE_TRACKBALL          =     1280 (&H500)
    DI8DEVTYPEMOUSE_TRADITIONAL        =      512 (&H200)
    DI8DEVTYPEMOUSE_UNKNOWN            =      256 (&H100)
    DI8DEVTYPEREMOTE_UNKNOWN           =      512 (&H200)
    DI8DEVTYPESCREENPTR_LIGHTGUN       =      768 (&H300)
    DI8DEVTYPESCREENPTR_LIGHTPEN       =     1024 (&H400)
    DI8DEVTYPESCREENPTR_TOUCH          =     1280 (&H500)
    DI8DEVTYPESCREENPTR_UNKNOWN        =      512 (&H200)
    DI8DEVTYPESUPPLEMENTAL_2NDHANDCONTROLLER = 768 (&H300)
    DI8DEVTYPESUPPLEMENTAL_COMBINEDPEDALS  = 2560 (&HA00)
    DI8DEVTYPESUPPLEMENTAL_DUALPEDALS      = 2816 (&HB00)
    DI8DEVTYPESUPPLEMENTAL_HANDTRACKER     = 1280 (&H500)
    DI8DEVTYPESUPPLEMENTAL_HEADTRACKER     = 1024 (&H400)
    DI8DEVTYPESUPPLEMENTAL_RUDDERPEDALS    = 3328 (&HD00)
    DI8DEVTYPESUPPLEMENTAL_SHIFTER         = 1792 (&H700)
    DI8DEVTYPESUPPLEMENTAL_SHIFTSTICKGATE  = 1536 (&H600)
    DI8DEVTYPESUPPLEMENTAL_SPLITTHROTTLE   = 2304 (&H900)
    DI8DEVTYPESUPPLEMENTAL_THREEPEDALS     = 3072 (&HC00)
    DI8DEVTYPESUPPLEMENTAL_THROTTLE        = 2048 (&H800)
    DI8DEVTYPESUPPLEMENTAL_UNKNOWN         =  512 (&H200)
    DIDEVTYPE_HID                      = 65536 (&H10000)
End Enum
```

Constants

DI8DEVTYPEJOYSTICK_ENHANCED
: Joystick that has more axes or point-of-view controllers than a standard joystick.

DI8DEVTYPE1STPERSON_LIMITED
: Device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPE1STPERSON_SHOOTER
: Device designed for first-person shooter games.

DI8DEVTYPE1STPERSON_SIXDOF
: Device with six degrees of freedom; that is, three lateral axes and three rotational axes.

DI8DEVTYPE1STPERSON_UNKNOWN
Unknown subtype.

DI8DEVTYPEDEVICECTRL_COMMSSELECTION
Control used to make communications selections.

DI8DEVTYPEDEVICECTRL_HARDWIRED
Device that must use its default configuration and cannot be remapped.

DI8DEVTYPEDEVICECTRL_UNKNOWN
Unknown subtype.

DI8DEVTYPEDRIVING_COMBINEDPEDALS
Steering device that reports acceleration and brake pedal values from a single axis.

DI8DEVTYPEDRIVING_DUALPEDALS
Steering device that reports acceleration and brake pedal values from separate axes.

DI8DEVTYPEDRIVING_HANDHELD
Hand-held steering device.

DI8DEVTYPEDRIVING_LIMITED
Steering device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEDRIVING_THREEPEDALS
Steering device that reports acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPEFLIGHT_LIMITED
Flight controller that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEFLIGHT_RC
Flight device based on a remote control for model aircraft.

DI8DEVTYPEFLIGHT_STICK
Joystick.

DI8DEVTYPEFLIGHT_YOKE
Yoke.

DI8DEVTYPEGAMEPAD_LIMITED
Gamepad that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_STANDARD
Standard gamepad that provides the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_TILT
Gamepad that can report x-axis and y-axis data based on the attitude of the controller.

DI8DEVTYPEJOYSTICK_LIMITED
Joystick that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEJOYSTICK_STANDARD
Standard joystick that provides the minimum number of device objects for action mapping.

DI8DEVTYPEKEYBOARD_J3100
Japanese J3100 keyboard.

DI8DEVTYPEKEYBOARD_JAPAN106
Japanese 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPANAX
Japanese AX keyboard.

DI8DEVTYPEKEYBOARD_NEC98
Japanese NEC PC98 keyboard.

DI8DEVTYPEKEYBOARD_NEC98106
Japanese NEC PC98 106-key keyboard.

DI8DEVTYPEKEYBOARD_NEC98LAPTOP
Japanese NEC PC98 laptop keyboard.

DI8DEVTYPEKEYBOARD_NOKIA1050
Nokia 1050 keyboard.

DI8DEVTYPEKEYBOARD_NOKIA9140
Nokia 9140 keyboard.

DI8DEVTYPEKEYBOARD_OLIVETTI
Olivetti 102-key keyboard.

DI8DEVTYPEKEYBOARD_PCAT
IBM PC/AT 84-key keyboard.

DI8DEVTYPEKEYBOARD_PCENH
IBM PC Enhanced 101/102-key or Microsoft® Natural® keyboard.

DI8DEVTYPEKEYBOARD_PCXT
IBM PC/XT 83-key keyboard.

DI8DEVTYPEKEYBOARD_UNKNOWN
Unknown subtype.

DI8DEVTYPEMOUSE_ABSOLUTE
Mouse that returns absolute axis data.

DI8DEVTYPEMOUSE_FINGERSTICK
Fingerstick.

DI8DEVTYPEMOUSE_TOUCHPAD
Touchpad.

DI8DEVTYPEMOUSE_TRACKBALL
Trackball.

DI8DEVTYPEMOUSE_TRADITIONAL
Traditional mouse.

DI8DEVTYPEMOUSE_UNKNOWN
Unknown subtype.

DI8DEVTYPEREMOTE_UNKNOWN

Remote-control device. This is the only defined remote-control device subtype.

DI8DEVTYPESCREENPTR_LIGHTGUN
Light gun.

DI8DEVTYPESCREENPTR_LIGHTPEN
Light pen.

DI8DEVTYPESCREENPTR_TOUCH
Touch screen.

DI8DEVTYPESCREENPTR_UNKNOWN
Unknown subtype.

DI8DEVTYPESUPPLEMENTAL_2NDHANDCONTROLLER
Secondary hand-held controller.

DI8DEVTYPESUPPLEMENTAL_COMBINEDPEDALS
Device whose primary function is to report acceleration and brake pedal values from a single axis.

DI8DEVTYPESUPPLEMENTAL_DUALPEDALS
Device whose primary function is to report acceleration and brake pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_HANDTRACKER
Device that tracks hand movement.

DI8DEVTYPESUPPLEMENTAL_HEADTRACKER
Device that tracks head movement.

DI8DEVTYPESUPPLEMENTAL_RUDDERPEDALS
Device with rudder pedals.

DI8DEVTYPESUPPLEMENTAL_SHIFTER
Device that reports gear selection from an axis.

DI8DEVTYPESUPPLEMENTAL_SHIFTSTICKGATE
Device that reports gear selection from button states.

DI8DEVTYPESUPPLEMENTAL_SPLITTHROTTLE
Device whose primary function is to report at least two throttle values. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_THREEPEDALS
Device whose primary function is to report acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_THROTTLE
Device whose primary function is to report a single throttle value. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_UNKNOWN
Unknown subtype.

DIDEVTYPE_HID
Human Interface Device (HID).

See Also

CONST_DI8DEVICETYPE

CONST_DI8DEVICETYPE

#Used to identify input device classes and types. One of these constants is passed as the *deviceType* parameter to the **DirectInput8.GetDIDevices** method to specify the class or type to enumerate. A packed value representing the device type and subtype is returned by the **DirectInputDeviceInstance8.GetDevType** method and in the **lDevType** member of the **DIDEVCAPS** type returned by **DirectInputDevice8.GetCapabilities**.

```
Enum CONST_DI8DEVICETYPE
   DI8DEVCLASS_ALL         =  0
   DI8DEVCLASS_DEVICE      =  1
   DI8DEVCLASS_GAMECTRL    =  4
   DI8DEVCLASS_KEYBOARD    =  3
   DI8DEVCLASS_POINTER     =  2
   DI8DEVTYPE_1STPERSON    = 24 (&H18)
   DI8DEVTYPE_DEVICE       = 17 (&H11)
   DI8DEVTYPE_DEVICECTRL   = 25 (&H19)
   DI8DEVTYPE_DRIVING      = 22 (&H16)
   DI8DEVTYPE_FLIGHT       = 23 (&H17)
   DI8DEVTYPE_GAMEPAD      = 21 (&H15)
   DI8DEVTYPE_JOYSTICK     = 20 (&H14)
   DI8DEVTYPE_KEYBOARD     = 19 (&H13)
   DI8DEVTYPE_MOUSE        = 18 (&H12)
   DI8DEVTYPE_REMOTE       = 27 (&H1B)
   DI8DEVTYPE_SCREENPOINTER = 26 (&H1A)
   DI8DEVTYPE_SUPPLEMENTAL  = 28 (&H1C)
End Enum
```

Constants

DI8DEVCLASS_ALL
All devices.

DI8DEVCLASS_DEVICE
All devices that do not fall into another class.

DI8DEVCLASS_GAMECTRL
All game controllers.

DI8DEVCLASS_KEYBOARD
All keyboards. Equivalent to DI8DEVTYPE_KEYBOARD.

DI8DEVCLASS_POINTER

IDH_CONST_DI8DEVICETYPE_dinput_vb

All devices of type DI8DEVTYPE_MOUSE and DI8DEVTYPE_SCREENPOINTER.

DI8DEVTYPE_1STPERSON
First-person action game device.

DI8DEVTYPE_DEVICE
Device that does not fall into another category.

DI8DEVTYPE_DEVICECTRL
Input device used to control another type of device from within the context of the application.

DI8DEVTYPE_DRIVING
Device for steering.

DI8DEVTYPE_FLIGHT
Controller for flight simulation.

DI8DEVTYPE_GAMEPAD
Gamepad.

DI8DEVTYPE_JOYSTICK
Joystick.

DI8DEVTYPE_KEYBOARD
Keyboard or keyboard-like device.

DI8DEVTYPE_MOUSE
A mouse or mouse-like device (such as a trackball).

DI8DEVTYPE_REMOTE
Remote-control device.

DI8DEVTYPE_SCREENPOINTER
Screen pointer.

DI8DEVTYPE_SUPPLEMENTAL
Specialized device with functionality unsuitable for the main control of an application, such as pedals used with a wheel.

See Also

CONST_DI8DEVICESUBTYPE

CONST_DIAFLAGS

#Used in **lFlags** member of the **DIACTION** type to specify how actions are to be mapped.

```
Enum CONST_DIAFLAGS
    DIA_APPMAPPED     = 2
    DIA_APPNOMAP      = 4
    DIA_FORCEFEEDBACK = 1
    DIA_NORANGE       = 8
```

IDH_CONST_DIAFLAGS_dinput_vb

```
End Enum
```

Constants

DIA_APPMAPPED
: The **lObjID** member of the **DIACTION** type is valid, and **DirectInputDevice8.BuildActionMap** should not override the application-defined mapping.

DIA_APPNOMAP
: This action is not to be mapped.

DIA_FORCEFEEDBACK
: The action must be mapped to an actuator or trigger.

DIA_NORANGE
: The default range is not to be set for this action. This flag can be set only for absolute axis actions.

CONST_DIAHFLAGS

#Used in **lHow** member of the **DIACTION** type to report how actions were mapped.

```
Enum CONST_DIAHFLAGS
   DIAH_APPREQUESTED =          2
   DIAH_DEFAULT      =         32 (&H20)
   DIAH_ERROR        = -2147483648 (&H80000000)
   DIAH_HWAPP        =          4
   DIAH_HWDEFAULT    =          8
   DIAH_UNMAPPED     =          0
   DIAH_USERCONFIG   =          1
End Enum
```

Constants

DIAH_APPREQUESTED
: The mapping was configured by the application, which specified the device (**guidInstance**) and device object (**lObjID**) when calling **DirectInputDevice8.BuildActionMap**.

DIAH_DEFAULT
: The mapping was determined by Microsoft® DirectInput® in the absence of other mapping information.

DIAH_ERROR
: An error occurred. The action cannot be matched to a control on the device. The action will be ignored when the action map is set.

DIAH_HWAPP
: The mapping was specified by the hardware manufacturer for this game.

IDH_CONST_DIAHFLAGS_dinput_vb

DIAH_HWDEFAULT
The mapping was specified by the hardware manufacturer for this genre.

DIAH_UNMAPPED
No suitable device object was found.

DIAH_USERCONFIG
The mapping was configured by the user.

CONST_DICDFLAGS

#Used in the *flags* parameter of **DirectInput8.ConfigureDevices** to specify the mode for the property sheet.

```
Enum CONST_DICDFLAGS
   DICD_DEFAULT = 0
   DICD_EDIT    = 1
End Enum
```

Constants

DICD_DEFAULT
Open the property sheet in view-only mode.

DICD_EDIT
Open the property sheet in edit mode. This mode enables the user to change action-to-control mappings. After the call returns, the application should assume current devices are no longer valid, release all device interfaces, and reinitialize them by calling **DirectInput8.GetDevicesBySemantics**.

CONST_DICOMMONDATAFORMATS

#Specify the data format in the *format* parameter of the **DirectInputDevice8.SetCommonDataFormat** method.

```
Enum CONST_DICOMMONDATAFORMATS
   DIFORMAT_JOYSTICK  =  3
   DIFORMAT_JOYSTICK2 =  4
   DIFORMAT_KEYBOARD  =  1
   DIFORMAT_MOUSE     =  2
   DIFORMAT_MOUSE2    =  5
End Enum
```

Constants

IDH_CONST_DICDFLAGS_dinput_vb
IDH_CONST_DICOMMONDATAFORMATS_dinput_vb

DIFORMAT_JOYSTICK
Joystick whose state data can be received in a **DIJOYSTATE** type.

DIFORMAT_JOYSTICK2
Joystick with extended capabilities whose state data can be received in a **DIJOYSTATE2** type.

DIFORMAT_KEYBOARD
Keyboard whose state data can be received in a **DIKEYBOARDSTATE** type.

DIFORMAT_MOUSE
Mouse whose state data can be received in a **DIMOUSESTATE** type.

DIFORMAT_MOUSE2
Mouse whose state data can be received in a **DIMOUSESTATE2** type.

CONST_DICONDITIONFLAGS

#Used in the **conditionFlags** member of the **DIEFFECT** type to set the axes and direction of a force-feedback effect.

```
Enum CONST_DICONDITIONFLAGS
   DICONDITION_USE_BOTH_AXES = 1
   DICONDITION_USE_DIRECTION = 2
End Enum
```

Constants

DICONDITION_USE_BOTH_AXES
Use data from both the **conditionX** and **conditionY** members of the **DIEFFECT** type

DICONDITION_USE_DIRECTION
Use data from only the **conditionX** member of the **DIEFFECT** type.

CONST_DIDALFLAGS

#Used to specify text alignment in the action-mapping property sheet.

```
Enum CONST_DIDALFLAGS
   DIDAL_BOTTOMALIGNED = 8
   DIDAL_CENTERED      = 0
   DIDAL_LEFTALIGNED   = 1
   DIDAL_MIDDLE        = 0
   DIDAL_RIGHTALIGNED  = 2
   DIDAL_TOPALIGNED    = 4
End Enum
```

IDH_CONST_DICONDITIONFLAGS_dinput_vb
IDH_CONST_DIDALFLAGS_dinput_vb

Constants

DIDAL_BOTTOMALIGNED
The text is aligned on the bottom border.

DIDAL_CENTERED
The text is horizontally centered.

DIDAL_LEFTALIGNED
The text is aligned on the left border.

DIDAL_MIDDLE
The text is vertically centered.

DIDAL_RIGHTALIGNED
The text is aligned on the right border.

DIDAL_TOPALIGNED
The text is aligned on the top border.

CONST_DIDATAFORMATFLAGS

#Used in the **lFlags** member of the **DIDATAFORMAT** type to describe additional attributes of the data format.

```
Enum CONST_DIDATAFORMATFLAGS
   DIDF_ABSAXIS = 1
   DIDF_RELAXIS = 2
End Enum
```

Constants

DIDF_ABSAXIS
The axes are in absolute mode. Setting this flag in the data format is equivalent to manually setting the axis mode property using the **DirectInputDevice8.SetProperty** method. This may not be combined with DIDF_RELAXIS flag.

DIDF_RELAXIS
The axes are in relative mode. Setting this flag in the data format is equivalent to manually setting the axis mode property using the **DirectInputDevice8.SetProperty** method. This may not be combined with the DIDF_ABSAXIS flag.

CONST_DIDBAMFLAGS

#Used in the **DirectInputDevice8.BuildActionMap** method.

```
Enum CONST_DIDBAMFLAGS
   DIDBAM_DEFAULT    = 0
```

IDH_CONST_DIDATAFORMATFLAGS_dinput_vb
IDH_CONST_DIDBAMFLAGS_dinput_vb

```
    DIDBAM_HWDEFAULTS = 4
    DIDBAM_INITIALIZE = 2
    DIDBAM_PRESERVE   = 1
End Enum
```

Constants

DIDBAM_DEFAULT
: Overwrite all mappings except application-specified mappings; that is, mappings that have the DIA_APPMAPPED flag in the **DIACTION** type.

DIDBAM_HWDEFAULTS
: Overwrite all mappings, including application-specified mappings. This flag is similar to DIDBAM_INITIALIZE, but automatically overrides user-mapped actions with the defaults specified by the device driver or Microsoft® DirectInput®.

DIDBAM_INITIALIZE
: Overwrite all mappings, including application-specified mappings.

DIDBAM_PRESERVE
: Preserve current mappings assigned for this device or any other configured device.

CONST_DIDEVCAPSFLAGS

[#]Used in the **lFlags** member of the **DIDEVCAPS** type to describe the Microsoft® DirectInput® device.

```
Enum CONST_DIDEVCAPSFLAGS
    DIDC_ALIAS            =  65536 (&H10000)
    DIDC_ATTACHED         =      1
    DIDC_DEADBAND         =  16384 (&H4000)
    DIDC_EMULATED         =      4
    DIDC_FFATTACK         =    512 (&H200)
    DIDC_FFFADE           =   1024 (&H400)
    DIDC_FORCEFEEDBACK    =    256 (&H100)
    DIDC_PHANTOM          = 131072 (&H20000)
    DIDC_POLLEDDATAFORMAT =      8
    DIDC_POLLEDDEVICE     =      2
    DIDC_POSNEGCOEFFICIENTS =   4096 (&H1000)
    DIDC_POSNEGSATURATION =   8192 (&H2000)
    DIDC_SATURATION       =   2048 (&H800)
End Enum
```

[#] IDH_CONST_DIDEVCAPSFLAGS_dinput_vb

Constants

DIDC_ALIAS
The device is a duplicate of another DirectInput device. Alias devices are by default not enumerated by **DirectInput8.GetDIDevices**.

DIDC_ATTACHED
The device is physically attached.

DIDC_DEADBAND
The device supports deadband for at least one force-feedback condition.

DIDC_EMULATED
If this flag is set, the data is coming from a user mode device interface—such as a Human Interface Device (HID)—or by some other ring-3 means. If it is not set, the data is coming directly from a kernel mode driver.

DIDC_FFATTACK
The force-feedback system supports the attack parameter for at least one effect. If the device does not support attack then the **lAttackLevel** and **lAttackTime** members of the **DIENVELOPE** type will be ignored by the device.

DIDC_FFFADE
The force-feedback system supports the fade parameter for at least one effect. If the device does not support fade then the **lFadeLevel** and **lFadeTime** members of the **DIENVELOPE** type will be ignored by the device.

DIDC_FORCEFEEDBACK
The device supports force feedback.

DIDC_PHANTOM
The device does not exist. It is a placeholder for a device that might exist in the future. Phantom devices are by default not enumerated by **DirectInput8.GetDIDevices**.

DIDC_POLLEDDATAFORMAT
At least one object in the current data format is polled rather than interrupt-driven. For these objects, the application must explicitly call the **DirectInputDevice8.Poll** method in order to obtain data.

DIDC_POLLEDDEVICE
At least one object on the device is polled rather than interrupt-driven. For these objects, the application must explicitly call the **DirectInputDevice8.Poll** method in order to obtain data. HID devices may contain a mixture of polled and non-polled objects.

DIDC_POSNEGCOEFFICIENTS
The force-feedback system supports two coefficient values for conditions (one for the positive displacement of the axis and one for the negative displacement of the axis) for at least one condition. If the device does not support both coefficients, then the negative coefficient in the **DICONDITION** type will be ignored.

DIDC_POSNEGSATURATION
The force-feedback system supports a maximum saturation for both positive and negative force output for at least one condition. If the device does not support

both saturation values, then the negative saturation in the **DICONDITION** type will be ignored.

DIDC_SATURATION
The force-feedback system supports the saturation of condition effects for at least one condition. If the device does not support saturation, then the force generated by a condition is limited only by the maximum force that the device can generate.

See Also

CONST_DIEFTFLAGS

CONST_DIDEVICEOBJINSTANCEFLAGS

#Used to describe device object capabilities. Members of this enumeration are returned by the **DirectInputDeviceObjectInstance.GetFlags** method. They are also present in the **DIOBJECTDATAFORMAT** type passed to the **DirectInputDevice8.SetDataFormat** method

```
Enum CONST_DIDEVICEOBJINSTANCEFLAGS
   DIDOI_ASPECTACCEL     =   768 (&H300)
   DIDOI_ASPECTFORCE     =  1024 (&H400)
   DIDOI_ASPECTMASK      =  3840 (&HF00)
   DIDOI_ASPECTPOSITION  =   256 (&H100)
   DIDOI_ASPECTVELOCITY  =   512 (&H200)
   DIDOI_FFACTUATOR      =     1
   DIDOI_FFEFFECTTRIGGER =     2
   DIDOI_GUIDISUSAGE     =  4096 (&H1000)
   DIDOI_POLLED          = 32768 (&H8000)
End Enum
```

Constants

DIDOI_ASPECTACCEL
The object reports acceleration information.

DIDOI_ASPECTFORCE
The object reports force information.

DIDOI_ASPECTMASK
The bits that are used to report aspect information. An object can represent at most one aspect.

DIDOI_ASPECTPOSITION
The object reports position information.

DIDOI_ASPECTVELOCITY

IDH_CONST_DIDEVICEOBJINSTANCEFLAGS_dinput_vb

The object reports velocity information.

DIDOI_FFACTUATOR
The object can have force-feedback effects applied to it.

DIDOI_FFEFFECTTRIGGER
The object can trigger playback of force-feedback effects.

DIDOI_GUIDISUSAGE
DIOBJECTDATAFORMAT.strGuid contains Human Interface Device (HID) usage and usage page in packed form.

DIDOI_POLLED
The object does not return data until the **DirectInputDevice8.Poll** method is called.

CONST_DIDFTFLAGS

#Used in the *flags* parameter of the **DirectInputDevice8.GetDeviceObjectsEnum** method to specify the type of device object to enumerate. These values are also returned by the **DirectInputDeviceObjectInstance.GetFlags** method to describe capabilities of the device object.

```
Enum CONST_DIDFTFLAGS
   DIDFT_ABSAXIS        =        2
   DIDFT_ALL            =        0
   DIDFT_ANYINSTANCE    =  16776960 (&HFFFF00)
   DIDFT_AXIS           =        3
   DIDFT_BUTTON         =       12
   DIDFT_COLLECTION     =       64 (&H40)
   DIDFT_FFACTUATOR     =  16777216 (&H1000000)
   DIDFT_FFEFFECTTRIGGER =  33554432 (&H2000000)
   DIDFT_INSTANCEMASK   =  16776960 (&HFFFF00)
   DIDFT_NOCOLLECTION   =  16776960 (&HFFFF00)
   DIDFT_NODATA         =      128 (&H80)
   DIDFT_OUTPUT         = 268435456 (&H10000000)
   DIDFT_POV            =       16 (&H10)
   DIDFT_PSHBUTTON      =        4
   DIDFT_RELAXIS        =        1
   DIDFT_TGLBUTTON      =        8
End Enum
```

Constants

DIDFT_ABSAXIS
Absolute axis.

DIDFT_ALIAS

IDH_CONST_DIDFTFLAGS_dinput_vb

Controls identified by a Human Interface Device (HID) usage alias. This flag applies only to HID-compliant USB devices.

DIDFT_ALL
All objects.

DIDFT_ANYINSTANCE
Use the first instance enumerated that meets other criteria such as DIDFT_AXIS or DIDFT_BUTTON.

DIDFT_AXIS
Axis, either absolute or relative.

DIDFT_BUTTON
Push button or a toggle button.

DIDFT_COLLECTION
Human Interface Device (HID) link collection. HID link collections do not generate data of their own.

DIDFT_FFACTUATOR
Object that contains a force-feedback actuator. In other words, forces can be applied to this object.

DIDFT_FFEFFECTTRIGGER
Object that can be used to trigger force-feedback effects.

DIDFT_INSTANCEMASK
Same as DIDFT_ANYINSTANCE.

DIDFT_NOCOLLECTION
Object that does not belong to any HID link collection; in other words, an object for which **DirectInputDeviceObjectInstance.GetCollectionNumber** returns 0.

DIDFT_NODATA
Object that does not generate data.

DIDFT_OUTPUT
Object to which data can be sent by using the **DirectInputDevice8.SendDeviceData** method.

DIDFT_POV
Point-of-view controller.

DIDFT_PSHBUTTON
Push button. A push button is reported as down when the user presses it and as up when the user releases it.

DIDFT_RELAXIS
Relative axis.

DIDFT_TGLBUTTON
Toggle button. A toggle button is reported as down when the user presses it and remains so until the user presses the button a second time.

DIDFT_VENDORDEFINED
An object of a type defined by the manufacturer.

CONST_DIDGDDFLAGS

[#]Used in the *flags* parameter of the **DirectInputDevice8.GetDeviceData** method to control the manner in which data is obtained.

```
Enum CONST_DIDGDDFLAGS
   DIGDD_DEFAULT = 0
   DIGDD_PEEK    = 1
End Enum
```

Constants

DIGDD_DEFAULT
: Remove retrieved items from the buffer.

DIGDD_PEEK
: Do not remove retrieved items from the buffer. A subsequent **GetDeviceData** call will read the same data.

CONST_DIDIFTFLAGS

[#]

Used in the *flags* member of the DIDEVICEIMAGEINFO type to specify the type of the file.

```
Enum CONST_DIDIFTFLAGS
   DIDIFT_CONFIGURATION =    1
   DIDIFT_OVERLAY       =    2
End Enum
```

Constants

DIDIFT_CONFIGURATION
: The file is used to display the current configuration of actions on the device. Overlay image coordinates are relative to the upper-left corner of the configuration image.

DIDIFT_OVERLAY
: The file is an overlay for a configuration image. The **OverlayOffset**, **OverlayRect**, **ObjID**, **CalloutLine**, **CalloutRect**, and **TextAlign** members are valid and contain data used to display the overlay and callout information for a single control on the device.

[#] IDH_CONST_DIDGDDFLAGS_dinput_vb
[#] IDH_CONST_DIDIFTFLAGS_dinput_vb

CONST_DIDSAMFLAGS

[#]Used in the **DirectInputDevice8.SetActionMap** method.

```
Enum CONST_DIDSAMFLAGS
   DIDSAM_DEFAULT   = 0
   DIDSAM_NOUSER    = 1
   DIDSAM_FORCESAVE = 2
End Enum
```

Constants

DIDSAM_DEFAULT
: Set the action map for this user. If the map differs from the current map, the new settings are saved to disk.

DIDSAM_FORCESAVE
: Always save the configuration to disk.

DIDSAM_NOUSER
: Reset user ownership for this device in the default configuration property sheet. Resetting user ownership does not remove the current action map.

CONST_DIEDBSFLFLAGS

[#]Used in the **DirectInput8.GetDevicesBySemantics** method to specify the scope of the enumeration.

```
Enum CONST_DIEDBSFLFLAGS
   DIEDBSFL_ATTACHEDONLY      =    0
   DIEDBSFL_AVAILABLEDEVICES  = 4096 (&H1000)
   DIEDBSFL_FORCEFEEDBACK     =  256 (&H100)
   DIEDBSFL_MULTIMICEKEYBOARDS = 8192 (&H2000)
   DIEDBSFL_NONGAMINGDEVICES  = 16384 (&H4000)
   DIEDBSFL_THISUSER          =   16 (&H10)
   DIEDBSFL_VALID             = 4368 (&H1110)
End Enum
```

Constants

DIEDBSFL_ATTACHEDONLY
: Only attached and installed devices are enumerated.

DIEDBSFL_AVAILABLEDEVICES
: Only unowned, installed devices are enumerated.

DIEDBSFL_FORCEFEEDBACK

IDH_CONST_DIDSAMFLAGS_dinput_vb
IDH_CONST_DIEDBSFLFLAGS_dinput_vb

Only devices that support force feedback are enumerated.

DIEDBSFL_MULTIMICEKEYBOARDS
Only secondary (non-system) keyboard and mouse devices.

DIEDBSFL_NONGAMINGDEVICES
Only HID-compliant devices whose primary purpose is not as a gaming device. Devices such as USB speakers and multimedia buttons on some keyboards would fall within this value.

DIEDBSFL_THISUSER
All installed devices for the user identified by *str1*, and all unowned devices, are enumerated.

DIEDBSFL_VALID
Not used by applications.

CONST_DIEFFFLAGS

#Used in the **lFlags** member of the **DIEFFECT** type.

```
Enum CONST_DIEFFFLAGS
   DIEFF_CARTESIAN     = 16 (&H10)
   DIEFF_OBJECTOFFSETS =  2
   DIEFF_POLAR         = 32 (&H20)
End Enum
```

Constants

DIEFF_CARTESIAN
The direction of the effect is given in Cartesian coordinates. **DIEFFECT.x** and **DIEFFECT.y** contain valid values.

DIEFF_OBJECTOFFSETS
The value of the **lTriggerButton** member of **DIEFFECT** is the offset of the button in the data structure for the device.

DIEFF_POLAR
The direction of the effect is given in polar coordinates. **DIEFFECT.x** contains a valid value.

Remarks

The default behavior of **DirectInputDevice8.CreateEffect** and **DirectInputDevice8.CreateCustomEffect** is to create the effect as if **DIEFFECT.lFlags** contained (DIEFF_OBJECTOFFSETS **Or** DIEFF_POLAR). In other words, it is not necessary to specify these flags.

IDH_CONST_DIEFFFLAGS_dinput_vb

CONST_DIEFTFLAGS

#Used to describe types and capabilities of force-feedback effects. These flags restrict effect enumeration in the **DirectInputDevice8.GetEffectsEnum** method to one or more of the primary types. They are also returned by the **DirectInputEnumEffects.GetType** method.

```
Enum CONST_DIEFTFLAGS
   DIEFT_ALL             =     0
   DIEFT_CONDITION       =     4
   DIEFT_CONSTANTFORCE   =     1
   DIEFT_CUSTOMFORCE     =     5
   DIEFT_DEADBAND        = 16384 (&H4000)
   DIEFT_FFATTACK        =   512 (&H200)
   DIEFT_FFFADE          =  1024 (&H400)
   DIEFT_HARDWARE        =   255 (&HFF)
   DIEFT_PERIODIC        =     3
   DIEFT_POSNEGCOEFFICIENTS =  4096 (&H1000)
   DIEFT_POSNEGSATURATION   =  8192 (&H2000)
   DIEFT_RAMPFORCE       =     2
   DIEFT_SATURATION      =  2048 (&H800)
End Enum
```

Constants

DIEFT_ALL
: All effects are to be enumerated.

DIEFT_CONDITION
: The effect is a condition, or conditions are to be enumerated.

DIEFT_CONSTANTFORCE
: The effect is a constant force, or constant forces are to be enumerated.

DIEFT_CUSTOMFORCE
: The effect is a custom force, or custom forces are to be enumerated.

DIEFT_DEADBAND
: The effect supports deadband.

DIEFT_FFATTACK
: The effect supports attack.

DIEFT_FFFADE
: The effect supports fade.

DIEFT_HARDWARE
: The effect is specific to the hardware, or hardware effects are to be enumerated.

DIEFT_PERIODIC
: The effect is periodic, or periodic effects are to be enumerated.

IDH_CONST_DIEFTFLAGS_dinput_vb

DIEFT_POSNEGCOEFFICIENTS
The condition supports different positive and negative coefficients.

DIEFT_POSNEGSATURATION
The effect supports different positive and negative saturation.

DIEFT_RAMPFORCE
The effect is a ramp force, or ramp forces are to be enumerated.

DIEFT_SATURATION
The effect supports saturation.

See Also

CONST_DIDEVCAPSFLAGS

CONST_DIEGESFLAGS

#Returned by the **DirectInputEffect.GetEffectStatus** method.

```
Enum CONST_DIEGESFLAGS
   DIEGES_EMULATED = 2
   DIEGES_PLAYING  = 1
End Enum
```

Constants

DIEGES_EMULATED
The effect is emulated.

DIEGES_PLAYING
The effect is playing.

CONST_DIENUMDEVICESFLAGS

#Used in the *flags* parameter of the **DirectInput8.GetDIDevices** method to indicate whether all devices or only attached devices are to be enumerated.

```
Enum CONST_DIENUMDEVICESFLAGS
   DIEDFL_ALLDEVICES      =      0
   DIEDFL_ATTACHEDONLY    =      1
   DIEDFL_FORCEFEEDBACK   =    256 (&H100)
   DIEDFL_INCLUDEALIASES  =  65536 (&H10000)
   DIEDFL_INCLUDEHIDDEN   = 262144 (&H40000)
   DIEDFL_INCLUDEPHANTOMS = 131072 (&H20000)
End Enum
```

IDH_CONST_DIEGESFLAGS_dinput_vb
IDH_CONST_DIENUMDEVICESFLAGS_dinput_vb

Constants

DIEDFL_ALLDEVICES
All installed devices. This is the default behavior.

DIEDFL_ATTACHEDONLY
Only attached and installed devices.

DIEDFL_FORCEFEEDBACK
Only devices that support force feedback.

DIEDFL_INCLUDEALIASES
Include devices that are aliases for other devices.

DIEDFL_INCLUDEHIDDEN
Include hidden devices. For more information about hidden devices, see **CONST_DIDEVCAPSFLAGS**.

DIEDFL_INCLUDEPHANTOMS
Include phantom (placeholder) devices.

CONST_DIEPFLAGS

#Miscellaneous flags for effects. Some members are passed to the **DirectInputEffect.SetParameters** method to specify which parameters are being set and the subsequent behavior of the effect. Some members are returned by the **DirectInputEnumEffects.GetStaticParams** and **DirectInputEnumEffects.GetDynamicParams** methods to describe support for the parameter.

```
Enum CONST_DIEPFLAGS
   DIEP_ALLPARAMS           =       1023 (&H3FF)
   DIEP_AXES                =         32 (&H20)
   DIEP_DIRECTION           =         64 (&H40)
   DIEP_DURATION            =          1
   DIEP_ENVELOPE            =        128 (&H80)
   DIEP_GAIN                =          4
   DIEP_NODOWNLOAD          = -2147483648 (&H80000000)
   DIEP_NORESTART           =  1073741824 (&H40000000)
   DIEP_SAMPLEPERIOD        =          2
   DIEP_START               =  536870912 (&H20000000)
   DIEP_STARTDELAY          =        512 (&H200)
   DIEP_TRIGGERBUTTON       =          8
   DIEP_TRIGGERREPEATINTERVAL =        16 (&H10)
   DIEP_TYPESPECIFICPARAMS  =        256 (&H100)
End Enum
```

IDH_CONST_DIEPFLAGS_dinput_vb

Constants

DIEP_ALLPARAMS
Not used.

DIEP_AXES
Not used

DIEP_DIRECTION
The **x** and **y** members of **DIEFFECT** are valid, or a direction parameter is supported. If you are setting or requesting effect parameters, you can specify that the direction is supplied or is to be returned in either Cartesian or polar coordinates by specifying DIEFF_CARTESIAN or DIEFF_POLAR in **DIEFFECT.lFlags**.

DIEP_DURATION
The **lDuration** member of **DIEFFECT** is valid, or the effect supports changing of the duration.

DIEP_ENVELOPE
The **envelope** member of **DIEFFECT** is valid, or the effect supports the application of an envelope.

DIEP_GAIN
The **lGain** member of **DIEFFECT** is valid, or the effect supports the application of gain.

DIEP_NODOWNLOAD
After setting parameters, the effect is not to be downloaded.

DIEP_NORESTART
Suppress the stopping and restarting of the effect in order to change parameters.

DIEP_SAMPLEPERIOD
The **lSamplePeriod** member of **DIEFFECT** is valid.

DIEP_START
Start the effect immediately.

DIEP_STARTDELAY
The **lStartDelay** member of **DIEFFECT** is valid, or the effect supports delayed start.

DIEP_TRIGGERBUTTON
The **lTriggerButton** member of **DIEFFECT** is valid, or the effect supports a trigger button.

DIEP_TRIGGERREPEATINTERVAL
The **lTriggerRepeatInterval** member of **DIEFFECT** is valid, or the effect supports trigger repeat.

DIEP_TYPESPECIFICPARAMS
The effect has type-specific parameters that can be changed by the application. This value is returned by **DirectInputEnumEffects.GetStaticParams** for all standard effects and for hardware-specific effects with application-modifiable parameters. Microsoft® DirectX® for Microsoft Visual Basic® supports hardware effects only if they do not have this flag set.

CONST_DIESFLAGS

#Used in the *flags* parameter of **DirectInputEffect.Start** to control behavior of force-feedback effect playback.

```
Enum CONST_DIESFLAGS
   DIES_NODOWNLOAD = -2147483648 (&H80000000)
   DIES_SOLO       =          1
End Enum
```

Constants

DIES_NODOWNLOAD
: Do not automatically download the effect.

DIES_SOLO
: Stop all other effects before playing this effect. If this flag is omitted, the effect is mixed with effects already playing on the device.

CONST_DIFEFFLAGS

#Used in the flags member of the **DirectInputDevice8.CreateEffectFromFile** method.

```
Enum CONST_DIFEFFLAGS
   DIFEF_DEFAULT          =  0
   DIFEF_INCLUDENONSTANDARD =  1
   DIFEF_MODIFYIFNEEDED     = 16 (&H10)
End Enum
```

Constants

DIFEF_DEFAULT
: Do not create a nonstandard effect or modify the effect parameters.

DIFEF_INCLUDENONSTANDARD
: Create an effect type that is not predefined by Microsoft® DirectInput®. If this flag is not set, the method fails when called on such effects.

DIFEF_MODIFYIFNEEDED
: Instruct DirectInput® to modify the authored effect, if necessary, so that it plays on the current device. For example, by default, an effect authored for two axes does not play on a single-axis device. Setting this flag allows the effect to play on a single axis.

IDH_CONST_DIESFLAGS_dinput_vb
IDH_CONST_DIFEFFLAGS_dinput_vb

CONST_DIGENRE

#Used in the **lSemantic** member of the **DIACTION** type to specify virtual controls and in the **lGenre** member of **DIACTIONFORMAT** to specify the application genre for action mapping.

For further details on any of the constants in this enumeration, see the appropriate section of the Action Mapping Constants page.

```
Enum CONST_DIGENRE
  DIAXIS_2DCONTROL_INOUT          = 587301379 (&H23018203)
  DIAXIS_2DCONTROL_LATERAL        = 587235841 (&H23008201)
  DIAXIS_2DCONTROL_MOVE           = 587268610 (&H23010202)
  DIAXIS_2DCONTROL_ROTATEZ        = 587350532 (&H23024204)
  DIAXIS_3DCONTROL_INOUT          = 604078595 (&H24018203)
  DIAXIS_3DCONTROL_LATERAL        = 604013057 (&H24008201)
  DIAXIS_3DCONTROL_MOVE           = 604045826 (&H24010202)
  DIAXIS_3DCONTROL_ROTATEX        = 604193284 (&H24034204)
  DIAXIS_3DCONTROL_ROTATEY        = 604160517 (&H2402C205)
  DIAXIS_3DCONTROL_ROTATEZ        = 604127750 (&H24024206)
  DIAXIS_ARCADEP_LATERAL          = 570458625 (&H22008201)
  DIAXIS_ARCADEP_MOVE             = 570491394 (&H22010202)
  DIAXIS_ARCADES_LATERAL          = 553681409 (&H21008201)
  DIAXIS_ARCADES_MOVE             = 553714178 (&H21010202)
  DIAXIS_BASEBALLB_LATERAL        = 251691521 (&H0F008201)
  DIAXIS_BASEBALLB_MOVE           = 251724290 (&H0F010202)
  DIAXIS_BASEBALLF_LATERAL        = 285245953 (&H11008201)
  DIAXIS_BASEBALLF_MOVE           = 285278722 (&H11010202)
  DIAXIS_BASEBALLP_LATERAL        = 268468737 (&H10008201)
  DIAXIS_BASEBALLP_MOVE           = 268501506 (&H10010202)
  DIAXIS_BBALLD_LATERAL           = 318800385 (&H13008201)
  DIAXIS_BBALLD_MOVE              = 318833154 (&H13010202)
  DIAXIS_BBALLO_LATERAL           = 302023169 (&H12008201)
  DIAXIS_BBALLO_MOVE              = 302055938 (&H12010202)
  DIAXIS_BIKINGM_BRAKE            = 470041091 (&H1C044203)
  DIAXIS_BIKINGM_PEDAL            = 469828098 (&H1C010202)
  DIAXIS_BIKINGM_TURN             = 469795329 (&H1C008201)
  DIAXIS_BROWSER_LATERAL          = 671121921 (&H28008201)
  DIAXIS_BROWSER_MOVE             = 671154690 (&H28010202)
  DIAXIS_BROWSER_VIEW             = 671187459 (&H28018203)
  DIAXIS_CADF_INOUT               = 620855811 (&H25018203)
  DIAXIS_CADF_LATERAL             = 620790273 (&H25008201)
  DIAXIS_CADF_MOVE                = 620823042 (&H25010202)
  DIAXIS_CADF_ROTATEX             = 620970500 (&H25034204)
  DIAXIS_CADF_ROTATEY             = 620937733 (&H2502C205)
```

IDH_CONST_DIGENRE_dinput_vb

DIAXIS_CADF_ROTATEZ = 620904966 (&H25024206)
DIAXIS_CADM_INOUT = 637633027 (&H26018203)
DIAXIS_CADM_LATERAL = 637567489 (&H26008201)
DIAXIS_CADM_MOVE = 637600258 (&H26010202)
DIAXIS_CADM_ROTATEX = 637747716 (&H26034204)
DIAXIS_CADM_ROTATEY = 637714949 (&H2602C205)
DIAXIS_CADM_ROTATEZ = 637682182 (&H26024206)
DIAXIS_DRIVINGC_ACCEL_AND_BRAKE = 33638916 (&H02014A04)
DIAXIS_DRIVINGC_ACCELERATE = 33788418 (&H02039202)
DIAXIS_DRIVINGC_BRAKE = 33821187 (&H02041203)
DIAXIS_DRIVINGC_STEER = 33589761 (&H03008A01)
DIAXIS_DRIVINGR_ACCEL_AND_BRAKE = 16861700 (&H01014A04)
DIAXIS_DRIVINGR_ACCELERATE = 17011202 (&H01039202)
DIAXIS_DRIVINGR_BRAKE = 17043971 (&H01041203)
DIAXIS_DRIVINGR_STEER = 16812545 (&H01008A01)
DIAXIS_DRIVINGT_ACCEL_AND_BRAKE = 50416133 (&H03014A05)
DIAXIS_DRIVINGT_ACCELERATE = 50565635 (&H03039203)
DIAXIS_DRIVINGT_BARREL = 50397698 (&H03010202)
DIAXIS_DRIVINGT_BRAKE = 50614788 (&H03045204)
DIAXIS_DRIVINGT_STEER = 50366977 (&H03008A01)
DIAXIS_FIGHTINGH_LATERAL = 134251009 (&H08008201)
DIAXIS_FIGHTINGH_MOVE = 134283778 (&H08010202)
DIAXIS_FIGHTINGH_ROTATE = 134365699 (&H08024203)
DIAXIS_FISHING_LATERAL = 234914305 (&H0E008201)
DIAXIS_FISHING_MOVE = 234947074 (&H0E010202)
DIAXIS_FISHING_ROTATE = 235028995 (&H0E024203)
DIAXIS_FLYINGC_BANK = 67144193 (&H04008A01)
DIAXIS_FLYINGC_BRAKE = 67398148 (&H04046A04)
DIAXIS_FLYINGC_FLAPS = 67459590 (&H04055A06)
DIAXIS_FLYINGC_PITCH = 67176962 (&H04010A02)
DIAXIS_FLYINGC_RUDDER = 67260933 (&H04025205)
DIAXIS_FLYINGC_THROTTLE = 67342851 (&H04039203)
DIAXIS_FLYINGH_BANK = 100698625 (&H06008A01)
DIAXIS_FLYINGH_COLLECTIVE = 100764163 (&H06018A03)
DIAXIS_FLYINGH_PITCH = 100731394 (&H06010A02)
DIAXIS_FLYINGH_THROTTLE = 100915717 (&H0603DA05)
DIAXIS_FLYINGH_TORQUE = 100817412 (&H06025A04)
DIAXIS_FLYINGM_BANK = 83921409 (&H05008A01)
DIAXIS_FLYINGM_BRAKE = 84173317 (&H05046205)
DIAXIS_FLYINGM_FLAPS = 84234758 (&H05055206)
DIAXIS_FLYINGM_PITCH = 83954178 (&H05010A02)
DIAXIS_FLYINGM_RUDDER = 84036100 (&H05024A04)
DIAXIS_FLYINGM_THROTTLE = 84120067 (&H05039203)
DIAXIS_FOOTBALLD_LATERAL = 385909249 (&H17008201)
DIAXIS_FOOTBALLD_MOVE = 385942018 (&H17010202)
DIAXIS_FOOTBALLO_LATERAL = 369132033 (&H16008201)

DIAXIS_FOOTBALLO_MOVE = 369164802 (&H16010202)
DIAXIS_FOOTBALLQ_LATERAL = 352354817 (&H15008201)
DIAXIS_FOOTBALLQ_MOVE = 352387586 (&H15010202)
DIAXIS_FPS_LOOKUPDOWN = 151093763 (&H09018203)
DIAXIS_FPS_MOVE = 151060994 (&H09010202)
DIAXIS_FPS_ROTATE = 151028225 (&H09008201)
DIAXIS_FPS_SIDESTEP = 151142916 (&H09024204)
DIAXIS_GOLF_LATERAL = 402686465 (&H18008201)
DIAXIS_GOLF_MOVE = 402719234 (&H18010202)
DIAXIS_HOCKEYD_LATERAL = 436240897 (&H1A008201)
DIAXIS_HOCKEYD_MOVE = 436273666 (&H1A010202)
DIAXIS_HOCKEYG_LATERAL = 453018113 (&H1B008201)
DIAXIS_HOCKEYG_MOVE = 453050882 (&H1B010202)
DIAXIS_HOCKEYO_LATERAL = 419463681 (&H19008201)
DIAXIS_HOCKEYO_MOVE = 419496450 (&H19010202)
DIAXIS_HUNTING_LATERAL = 218137089 (&H0D008201)
DIAXIS_HUNTING_MOVE = 218169858 (&H0D010202)
DIAXIS_HUNTING_ROTATE = 218251779 (&H0D024203)
DIAXIS_MECHA_ROTATE = 687997443 (&H29020203)
DIAXIS_MECHA_STEER = 687899137 (&H29008201)
DIAXIS_MECHA_THROTTLE = 688095748 (&H29038204)
DIAXIS_MECHA_TORSO = 687931906 (&H29010202)
DIAXIS_RACQUET_LATERAL = 536904193 (&H20008201)
DIAXIS_RACQUET_MOVE = 536936962 (&H20010202)
DIAXIS_REMOTE_SLIDER = 654639617 (&H27050201)
DIAXIS_REMOTE_SLIDER2 = 654656002 (&H27054202)
DIAXIS_SKIING_SPEED = 486605314 (&H1D010202)
DIAXIS_SKIING_TURN = 486572545 (&H1D008201)
DIAXIS_SOCCERD_LATERAL = 520126977 (&H1F008201)
DIAXIS_SOCCERD_MOVE = 520159746 (&H1F010202)
DIAXIS_SOCCERO_BEND = 503415299 (&H1E018203)
DIAXIS_SOCCERO_LATERAL = 503349761 (&H1E008201)
DIAXIS_SOCCERO_MOVE = 503382530 (&H1E010202)
DIAXIS_SPACESIM_CLIMB = 117555716 (&H0701C204)
DIAXIS_SPACESIM_LATERAL = 117473793 (&H07008201)
DIAXIS_SPACESIM_MOVE = 117506562 (&H07010202)
DIAXIS_SPACESIM_ROTATE = 117588485 (&H07024205)
DIAXIS_SPACESIM_THROTTLE = 117670403 (&H07038203)
DIAXIS_STRATEGYR_LATERAL = 184582657 (&H0B008201)
DIAXIS_STRATEGYR_MOVE = 184615426 (&H0B010202)
DIAXIS_STRATEGYR_ROTATE = 184697347 (&H0B024203)
DIAXIS_STRATEGYT_LATERAL = 201359873 (&H0C008201)
DIAXIS_STRATEGYT_MOVE = 201392642 (&H0C010202)
DIAXIS_TPS_MOVE = 167838210 (&H0A010202)
DIAXIS_TPS_STEP = 167821827 (&H0A00C203)
DIAXIS_TPS_TURN = 167903745 (&H0A020201)

```
DIBUTTON_2DCONTROL_DEVICE        = 587220222 (&H230044FE)
DIBUTTON_2DCONTROL_DISPLAY       = 587219973 (&H23004405)
DIBUTTON_2DCONTROL_MENU          = 587203837 (&H230004FD)
DIBUTTON_2DCONTROL_PAUSE         = 587220220 (&H230044FC)
DIBUTTON_2DCONTROL_SELECT        = 587203585 (&H23000401)
DIBUTTON_2DCONTROL_SPECIAL       = 587203587 (&H23000403)
DIBUTTON_2DCONTROL_SPECIAL1      = 587203586 (&H23000402)
DIBUTTON_2DCONTROL_SPECIAL2      = 587203588 (&H23000404)
DIBUTTON_3DCONTROL_DEVICE        = 603997438 (&H240044FE)
DIBUTTON_3DCONTROL_DISPLAY       = 603997189 (&H24004405)
DIBUTTON_3DCONTROL_MENU          = 603981053 (&H240004FD)
DIBUTTON_3DCONTROL_PAUSE         = 603997436 (&H240044FC)
DIBUTTON_3DCONTROL_SELECT        = 603980801 (&H24000401)
DIBUTTON_3DCONTROL_SPECIAL       = 603980803 (&H24000403)
DIBUTTON_3DCONTROL_SPECIAL1      = 603980802 (&H24000402)
DIBUTTON_3DCONTROL_SPECIAL2      = 603980804 (&H24000404)
DIBUTTON_ARCADEP_BACK_LINK       = 570508520 (&H220144E8)
DIBUTTON_ARCADEP_CROUCH          = 570426371 (&H22000403)
DIBUTTON_ARCADEP_DEVICE          = 570443006 (&H220044FE)
DIBUTTON_ARCADEP_FIRE            = 570426370 (&H22000402)
DIBUTTON_ARCADEP_FIRESECONDARY   = 570442758 (&H22004406)
DIBUTTON_ARCADEP_FORWARD_LINK    = 570508512 (&H220144E0)
DIBUTTON_ARCADEP_JUMP            = 570426369 (&H22000401)
DIBUTTON_ARCADEP_LEFT_LINK       = 570475748 (&H2200C4E4)
DIBUTTON_ARCADEP_MENU            = 570426621 (&H220004FD)
DIBUTTON_ARCADEP_PAUSE           = 570443004 (&H220044FC)
DIBUTTON_ARCADEP_RIGHT_LINK      = 570475756 (&H2200C4EC)
DIBUTTON_ARCADEP_SELECT          = 570426373 (&H22000405)
DIBUTTON_ARCADEP_SPECIAL         = 570426372 (&H22000404)
DIBUTTON_ARCADEP_VIEW_DOWN_LINK  = 570934504 (&H2207C4E8)
DIBUTTON_ARCADEP_VIEW_LEFT_LINK  = 570934500 (&H2207C4E4)
DIBUTTON_ARCADEP_VIEW_RIGHT_LINK = 570934508 (&H2207C4EC)
DIBUTTON_ARCADEP_VIEW_UP_LINK    = 570934496 (&H2207C4E0)
DIBUTTON_ARCADES_ATTACK          = 553649155 (&H21000403)
DIBUTTON_ARCADES_BACK_LINK       = 553731304 (&H210144E8)
DIBUTTON_ARCADES_CARRY           = 553649154 (&H21000402)
DIBUTTON_ARCADES_DEVICE          = 553665790 (&H210044FE)
DIBUTTON_ARCADES_FORWARD_LINK    = 553731296 (&H210144E0)
DIBUTTON_ARCADES_LEFT_LINK       = 553698532 (&H2100C4E4)
DIBUTTON_ARCADES_MENU            = 553649405 (&H210004FD)
DIBUTTON_ARCADES_PAUSE           = 553665788 (&H210044FC)
DIBUTTON_ARCADES_RIGHT_LINK      = 553698540 (&H2100C4EC)
DIBUTTON_ARCADES_SELECT          = 553649157 (&H21000405)
DIBUTTON_ARCADES_SPECIAL         = 553649156 (&H21000404)
DIBUTTON_ARCADES_THROW           = 553649153 (&H21000401)
DIBUTTON_ARCADES_VIEW_DOWN_LINK  = 554157288 (&H2107C4E8)
```

```
DIBUTTON_ARCADES_VIEW_LEFT_LINK   = 554157284 (&H2107C4E4)
DIBUTTON_ARCADES_VIEW_RIGHT_LINK  = 554157292 (&H2107C4EC)
DIBUTTON_ARCADES_VIEW_UP_LINK     = 554157280 (&H2107C4E0)
DIBUTTON_BASEBALLB_BACK_LINK      = 251741416 (&H0F0144E8)
DIBUTTON_BASEBALLB_BOX            = 251675658 (&H0F00440A)
DIBUTTON_BASEBALLB_BUNT           = 251659268 (&H0F000404)
DIBUTTON_BASEBALLB_BURST          = 251659270 (&H0F000406)
DIBUTTON_BASEBALLB_CONTACT        = 251659272 (&H0F000408)
DIBUTTON_BASEBALLB_DEVICE         = 251675902 (&H0F0044FE)
DIBUTTON_BASEBALLB_FORWARD_LINK   = 251741408 (&H0F0144E0)
DIBUTTON_BASEBALLB_LEFT_LINK      = 251708644 (&H0F00C4E4)
DIBUTTON_BASEBALLB_MENU           = 251659517 (&H0F0004FD)
DIBUTTON_BASEBALLB_NORMAL         = 251659266 (&H0F000402)
DIBUTTON_BASEBALLB_NOSTEAL        = 251675657 (&H0F004409)
DIBUTTON_BASEBALLB_PAUSE          = 251675900 (&H0F0044FC)
DIBUTTON_BASEBALLB_POWER          = 251659267 (&H0F000403)
DIBUTTON_BASEBALLB_RIGHT_LINK     = 251708652 (&H0F00C4EC)
DIBUTTON_BASEBALLB_SELECT         = 251659265 (&H0F000401)
DIBUTTON_BASEBALLB_SLIDE          = 251659271 (&H0F000407)
DIBUTTON_BASEBALLB_STEAL          = 251659269 (&H0F000405)
DIBUTTON_BASEBALLF_AIM_LEFT_LINK  = 285263076 (&H1100C4E4)
DIBUTTON_BASEBALLF_AIM_RIGHT_LINK = 285263084 (&H1100C4EC)
DIBUTTON_BASEBALLF_BACK_LINK      = 285295848 (&H110144E8)
DIBUTTON_BASEBALLF_BURST          = 285213700 (&H11000404)
DIBUTTON_BASEBALLF_DEVICE         = 285230334 (&H110044FE)
DIBUTTON_BASEBALLF_DIVE           = 285213702 (&H11000406)
DIBUTTON_BASEBALLF_FORWARD_LINK   = 285295840 (&H110144E0)
DIBUTTON_BASEBALLF_JUMP           = 285213701 (&H11000405)
DIBUTTON_BASEBALLF_MENU           = 285213949 (&H110004FD)
DIBUTTON_BASEBALLF_NEAREST        = 285213697 (&H11000401)
DIBUTTON_BASEBALLF_PAUSE          = 285230332 (&H110044FC)
DIBUTTON_BASEBALLF_SHIFTIN        = 285230087 (&H11004407)
DIBUTTON_BASEBALLF_SHIFTOUT       = 285230088 (&H11004408)
DIBUTTON_BASEBALLF_THROW1         = 285213698 (&H11000402)
DIBUTTON_BASEBALLF_THROW2         = 285213699 (&H11000403)
DIBUTTON_BASEBALLP_BACK_LINK      = 268518632 (&H100144E8)
DIBUTTON_BASEBALLP_BASE           = 268436483 (&H10000403)
DIBUTTON_BASEBALLP_DEVICE         = 268453118 (&H100044FE)
DIBUTTON_BASEBALLP_FAKE           = 268436485 (&H10000405)
DIBUTTON_BASEBALLP_FORWARD_LINK   = 268518624 (&H100144E0)
DIBUTTON_BASEBALLP_LEFT_LINK      = 268485860 (&H1000C4E4)
DIBUTTON_BASEBALLP_LOOK           = 268452871 (&H10004407)
DIBUTTON_BASEBALLP_MENU           = 268436733 (&H100004FD)
DIBUTTON_BASEBALLP_PAUSE          = 268453116 (&H100044FC)
DIBUTTON_BASEBALLP_PITCH          = 268436482 (&H10000402)
DIBUTTON_BASEBALLP_RIGHT_LINK     = 268485868 (&H1000C4EC)
```

DIBUTTON_BASEBALLP_SELECT = 268436481 (&H10000401)
DIBUTTON_BASEBALLP_THROW = 268436484 (&H10000404)
DIBUTTON_BASEBALLP_WALK = 268452870 (&H10004406)
DIBUTTON_BBALLD_BACK_LINK = 318850280 (&H130144E8)
DIBUTTON_BBALLD_BURST = 318768134 (&H13000406)
DIBUTTON_BBALLD_DEVICE = 318784766 (&H130044FE)
DIBUTTON_BBALLD_FAKE = 318768131 (&H13000403)
DIBUTTON_BBALLD_FORWARD_LINK = 318850272 (&H130144E0)
DIBUTTON_BBALLD_JUMP = 318768129 (&H13000401)
DIBUTTON_BBALLD_LEFT_LINK = 318817508 (&H1300C4E4)
DIBUTTON_BBALLD_MENU = 318768381 (&H130004FD)
DIBUTTON_BBALLD_PAUSE = 318784764 (&H130044FC)
DIBUTTON_BBALLD_PLAY = 318768135 (&H13000407)
DIBUTTON_BBALLD_PLAYER = 318768133 (&H13000405)
DIBUTTON_BBALLD_RIGHT_LINK = 318817516 (&H1300C4EC)
DIBUTTON_BBALLD_SPECIAL = 318768132 (&H13000404)
DIBUTTON_BBALLD_STEAL = 318768130 (&H13000402)
DIBUTTON_BBALLD_SUBSTITUTE = 318784521 (&H13004409)
DIBUTTON_BBALLD_TIMEOUT = 318784520 (&H13004408)
DIBUTTON_BBALLO_BACK_LINK = 302073064 (&H120144E8)
DIBUTTON_BBALLO_BURST = 301990919 (&H12000407)
DIBUTTON_BBALLO_CALL = 301990920 (&H12000408)
DIBUTTON_BBALLO_DEVICE = 302007550 (&H120044FE)
DIBUTTON_BBALLO_DUNK = 301990914 (&H12000402)
DIBUTTON_BBALLO_FAKE = 301990916 (&H12000404)
DIBUTTON_BBALLO_FORWARD_LINK = 302073056 (&H120144E0)
DIBUTTON_BBALLO_JAB = 302007307 (&H1200440B)
DIBUTTON_BBALLO_LEFT_LINK = 302040292 (&H1200C4E4)
DIBUTTON_BBALLO_MENU = 301991165 (&H120004FD)
DIBUTTON_BBALLO_PASS = 301990915 (&H12000403)
DIBUTTON_BBALLO_PAUSE = 302007548 (&H120044FC)
DIBUTTON_BBALLO_PLAY = 302007306 (&H1200440A)
DIBUTTON_BBALLO_PLAYER = 301990918 (&H12000406)
DIBUTTON_BBALLO_POST = 302007308 (&H1200440C)
DIBUTTON_BBALLO_RIGHT_LINK = 302040300 (&H1200C4EC)
DIBUTTON_BBALLO_SCREEN = 302007305 (&H12004409)
DIBUTTON_BBALLO_SHOOT = 301990913 (&H12000401)
DIBUTTON_BBALLO_SPECIAL = 301990917 (&H12000405)
DIBUTTON_BBALLO_SUBSTITUTE = 302007310 (&H1200440E)
DIBUTTON_BBALLO_TIMEOUT = 302007309 (&H1200440D)
DIBUTTON_BIKINGM_BRAKE_BUTTON_LINK= 470041832 (&H1C0444E8)
DIBUTTON_BIKINGM_CAMERA = 469763074 (&H1C000402)
DIBUTTON_BIKINGM_DEVICE = 469779710 (&H1C0044FE)
DIBUTTON_BIKINGM_FASTER_LINK = 469845216 (&H1C0144E0)
DIBUTTON_BIKINGM_JUMP = 469763073 (&H1C000401)
DIBUTTON_BIKINGM_LEFT_LINK = 469812452 (&H1C00C4E4)

DIBUTTON_BIKINGM_MENU = 469763325 (&H1C0004FD)
DIBUTTON_BIKINGM_PAUSE = 469779708 (&H1C0044FC)
DIBUTTON_BIKINGM_RIGHT_LINK = 469812460 (&H1C00C4EC)
DIBUTTON_BIKINGM_SELECT = 469763076 (&H1C000404)
DIBUTTON_BIKINGM_SLOWER_LINK = 469845224 (&H1C0144E8)
DIBUTTON_BIKINGM_SPECIAL1 = 469763075 (&H1C000403)
DIBUTTON_BIKINGM_SPECIAL2 = 469763077 (&H1C000405)
DIBUTTON_BIKINGM_ZOOM = 469779462 (&H1C004406)
DIBUTTON_BROWSER_DEVICE = 671106302 (&H280044FE)
DIBUTTON_BROWSER_FAVORITES = 671106054 (&H28004406)
DIBUTTON_BROWSER_HISTORY = 671106057 (&H28004409)
DIBUTTON_BROWSER_HOME = 671106053 (&H28004405)
DIBUTTON_BROWSER_MENU = 671089917 (&H280004FD)
DIBUTTON_BROWSER_NEXT = 671106055 (&H28004407)
DIBUTTON_BROWSER_PAUSE = 671106300 (&H280044FC)
DIBUTTON_BROWSER_PREVIOUS = 671106056 (&H28004408)
DIBUTTON_BROWSER_PRINT = 671106058 (&H2800440A)
DIBUTTON_BROWSER_REFRESH = 671089666 (&H28000402)
DIBUTTON_BROWSER_SEARCH = 671106051 (&H28004403)
DIBUTTON_BROWSER_SELECT = 671089665 (&H28000401)
DIBUTTON_BROWSER_STOP = 671106052 (&H28004404)
DIBUTTON_CADF_DEVICE = 620774654 (&H250044FE)
DIBUTTON_CADF_DISPLAY = 620774405 (&H25004405)
DIBUTTON_CADF_MENU = 620758269 (&H250004FD)
DIBUTTON_CADF_PAUSE = 620774652 (&H250044FC)
DIBUTTON_CADF_SELECT = 620758017 (&H25000401)
DIBUTTON_CADF_SPECIAL = 620758019 (&H25000403)
DIBUTTON_CADF_SPECIAL1 = 620758018 (&H25000402)
DIBUTTON_CADF_SPECIAL2 = 620758020 (&H25000404)
DIBUTTON_CADM_DEVICE = 637551870 (&H260044FE)
DIBUTTON_CADM_DISPLAY = 637551621 (&H26004405)
DIBUTTON_CADM_MENU = 637535485 (&H260004FD)
DIBUTTON_CADM_PAUSE = 637551868 (&H260044FC)
DIBUTTON_CADM_SELECT = 637535233 (&H26000401)
DIBUTTON_CADM_SPECIAL = 637535235 (&H26000403)
DIBUTTON_CADM_SPECIAL1 = 637535234 (&H26000402)
DIBUTTON_CADM_SPECIAL2 = 637535236 (&H26000404)
DIBUTTON_DRIVINGC_ACCELERATE_LINK = 33805536 (&H0203D4E0)
DIBUTTON_DRIVINGC_AIDS = 33571847 (&H02004407)
DIBUTTON_DRIVINGC_BRAKE = 33573896 (&H02004C08)
DIBUTTON_DRIVINGC_DASHBOARD = 33571846 (&H02004406)
DIBUTTON_DRIVINGC_DEVICE = 33572094 (&H020044FE)
DIBUTTON_DRIVINGC_FIRE = 33557505 (&H02000C01)
DIBUTTON_DRIVINGC_FIRESECONDARY = 33573897 (&H02004C09)
DIBUTTON_DRIVINGC_GLANCE_LEFT_LINK = 34063588 (&H0207C4E4)
DIBUTTON_DRIVINGC_GLANCE_RIGHT_LINK= 34063596 (&H0207C4EC)

DIBUTTON_DRIVINGC_MENU = 33555709 (&H020004FD)
DIBUTTON_DRIVINGC_PAUSE = 33572092 (&H020044FC)
DIBUTTON_DRIVINGC_SHIFTDOWN = 33573893 (&H02004C05)
DIBUTTON_DRIVINGC_SHIFTUP = 33573892 (&H02004C04)
DIBUTTON_DRIVINGC_STEER_LEFT_LINK = 33606884 (&H0200CCE4)
DIBUTTON_DRIVINGC_STEER_RIGHT_LINK = 33606892 (&H0200CCEC)
DIBUTTON_DRIVINGC_TARGET = 33557507 (&H02000C03)
DIBUTTON_DRIVINGC_WEAPONS = 33557506 (&H02000C02)
DIBUTTON_DRIVINGR_ACCELERATE_LINK = 17028320 (&H0103D4E0)
DIBUTTON_DRIVINGR_AIDS = 16794630 (&H01004406)
DIBUTTON_DRIVINGR_BOOST = 16794632 (&H01004408)
DIBUTTON_DRIVINGR_BRAKE = 16796676 (&H01004C04)
DIBUTTON_DRIVINGR_DASHBOARD = 16794629 (&H01004405)
DIBUTTON_DRIVINGR_DEVICE = 16794878 (&H010044FE)
DIBUTTON_DRIVINGR_GLANCE_LEFT_LINK = 17286372 (&H0107C4E4)
DIBUTTON_DRIVINGR_GLANCE_RIGHT_LINK= 17286380 (&H0107C4EC)
DIBUTTON_DRIVINGR_MAP = 16794631 (&H01004407)
DIBUTTON_DRIVINGR_MENU = 16778493 (&H010004FD)
DIBUTTON_DRIVINGR_PAUSE = 16794876 (&H010044FC)
DIBUTTON_DRIVINGR_PIT = 16794633 (&H01004409)
DIBUTTON_DRIVINGR_SHIFTDOWN = 16780290 (&H01000C02)
DIBUTTON_DRIVINGR_SHIFTUP = 16780289 (&H01000C01)
DIBUTTON_DRIVINGR_STEER_LEFT_LINK = 16829668 (&H0100CCE4)
DIBUTTON_DRIVINGR_STEER_RIGHT_LINK = 16829676 (&H0100CCEC)
DIBUTTON_DRIVINGR_VIEW = 16784387 (&H01001C03)
DIBUTTON_DRIVINGT_ACCELERATE_LINK = 50582752 (&H0303D4E0)
DIBUTTON_DRIVINGT_BARREL_DOWN_LINK = 50414824 (&H030144E8)
DIBUTTON_DRIVINGT_BARREL_UP_LINK = 50414816 (&H030144E0)
DIBUTTON_DRIVINGT_BRAKE = 50351110 (&H03004C06)
DIBUTTON_DRIVINGT_DASHBOARD = 50355205 (&H03005C05)
DIBUTTON_DRIVINGT_DEVICE = 50349310 (&H030044FE)
DIBUTTON_DRIVINGT_FIRE = 50334721 (&H03000C01)
DIBUTTON_DRIVINGT_FIRESECONDARY = 50351111 (&H03004C07)
DIBUTTON_DRIVINGT_GLANCE_LEFT_LINK = 50840804 (&H0307C4E4)
DIBUTTON_DRIVINGT_GLANCE_RIGHT_LINK= 50840812 (&H0307C4EC)
DIBUTTON_DRIVINGT_MENU = 50332925 (&H030004FD)
DIBUTTON_DRIVINGT_PAUSE = 50349308 (&H030044FC)
DIBUTTON_DRIVINGT_STEER_LEFT_LINK = 50384100 (&H0300CCE4)
DIBUTTON_DRIVINGT_STEER_RIGHT_LINK = 50384108 (&H0300CCEC)
DIBUTTON_DRIVINGT_TARGET = 50334723 (&H03000C03)
DIBUTTON_DRIVINGT_VIEW = 50355204 (&H03005C04)
DIBUTTON_DRIVINGT_WEAPONS = 50334722 (&H03000C02)
DIBUTTON_FIGHTINGH_BACKWARD_LINK = 134300904 (&H080144E8)
DIBUTTON_FIGHTINGH_BLOCK = 134218755 (&H08000403)
DIBUTTON_FIGHTINGH_CROUCH = 134218756 (&H08000404)
DIBUTTON_FIGHTINGH_DEVICE = 134235390 (&H080044FE)

DIBUTTON_FIGHTINGH_DISPLAY = 134235145 (&H08004409)
DIBUTTON_FIGHTINGH_DODGE = 134235146 (&H0800440A)
DIBUTTON_FIGHTINGH_FORWARD_LINK = 134300896 (&H080144E0)
DIBUTTON_FIGHTINGH_JUMP = 134218757 (&H08000405)
DIBUTTON_FIGHTINGH_KICK = 134218754 (&H08000402)
DIBUTTON_FIGHTINGH_LEFT_LINK = 134268132 (&H0800C4E4)
DIBUTTON_FIGHTINGH_MENU = 134219005 (&H080004FD)
DIBUTTON_FIGHTINGH_PAUSE = 134235388 (&H080044FC)
DIBUTTON_FIGHTINGH_PUNCH = 134218753 (&H08000401)
DIBUTTON_FIGHTINGH_RIGHT_LINK = 134268140 (&H0800C4EC)
DIBUTTON_FIGHTINGH_SELECT = 134235144 (&H08004408)
DIBUTTON_FIGHTINGH_SPECIAL1 = 134218758 (&H08000406)
DIBUTTON_FIGHTINGH_SPECIAL2 = 134218759 (&H08000407)
DIBUTTON_FISHING_BACK_LINK = 234964200 (&H0E0144E8)
DIBUTTON_FISHING_BAIT = 234882052 (&H0E000404)
DIBUTTON_FISHING_BINOCULAR = 234882051 (&H0E000403)
DIBUTTON_FISHING_CAST = 234882049 (&H0E000401)
DIBUTTON_FISHING_CROUCH = 234898439 (&H0E004407)
DIBUTTON_FISHING_DEVICE = 234898686 (&H0E0044FE)
DIBUTTON_FISHING_DISPLAY = 234898438 (&H0E004406)
DIBUTTON_FISHING_FORWARD_LINK = 234964192 (&H0E0144E0)
DIBUTTON_FISHING_JUMP = 234898440 (&H0E004408)
DIBUTTON_FISHING_LEFT_LINK = 234931428 (&H0E00C4E4)
DIBUTTON_FISHING_MAP = 234882053 (&H0E000405)
DIBUTTON_FISHING_MENU = 234882301 (&H0E0004FD)
DIBUTTON_FISHING_PAUSE = 234898684 (&H0E0044FC)
DIBUTTON_FISHING_RIGHT_LINK = 234931436 (&H0E00C4EC)
DIBUTTON_FISHING_ROTATE_LEFT_LINK = 235029732 (&H0E0244E4)
DIBUTTON_FISHING_ROTATE_RIGHT_LINK= 235029740 (&H0E0244EC)
DIBUTTON_FISHING_TYPE = 234882050 (&H0E000402)
DIBUTTON_FLYINGC_BRAKE_LINK = 67398880 (&H04046CE0)
DIBUTTON_FLYINGC_DEVICE = 67126526 (&H040044FE)
DIBUTTON_FLYINGC_DISPLAY = 67118082 (&H04002402)
DIBUTTON_FLYINGC_FASTER_LINK = 67359968 (&H0403D4E0)
DIBUTTON_FLYINGC_FLAPSDOWN = 67134469 (&H04006405)
DIBUTTON_FLYINGC_FLAPSUP = 67134468 (&H04006404)
DIBUTTON_FLYINGC_GEAR = 67120131 (&H04002C03)
DIBUTTON_FLYINGC_GLANCE_DOWN_LINK = 67618024 (&H0407C4E8)
DIBUTTON_FLYINGC_GLANCE_LEFT_LINK = 67618020 (&H0407C4E4)
DIBUTTON_FLYINGC_GLANCE_RIGHT_LINK = 67618028 (&H0407C4EC)
DIBUTTON_FLYINGC_GLANCE_UP_LINK = 67618016 (&H0407C4E0)
DIBUTTON_FLYINGC_MENU = 67110141 (&H040004FD)
DIBUTTON_FLYINGC_PAUSE = 67126524 (&H040044FC)
DIBUTTON_FLYINGC_SLOWER_LINK = 67359976 (&H0403D4E8)
DIBUTTON_FLYINGC_VIEW = 67118081 (&H04002401)
DIBUTTON_FLYINGH_COUNTER = 100684804 (&H06005404)

DIBUTTON_FLYINGH_DEVICE = 100680958 (&H060044FE)
DIBUTTON_FLYINGH_FASTER_LINK = 100916448 (&H0603DCE0)
DIBUTTON_FLYINGH_FIRE = 100668417 (&H06001401)
DIBUTTON_FLYINGH_FIRESECONDARY = 100682759 (&H06004C07)
DIBUTTON_FLYINGH_GEAR = 100688902 (&H06006406)
DIBUTTON_FLYINGH_GLANCE_DOWN_LINK = 101172456 (&H0607C4E8)
DIBUTTON_FLYINGH_GLANCE_LEFT_LINK = 101172452 (&H0607C4E4)
DIBUTTON_FLYINGH_GLANCE_RIGHT_LINK= 101172460 (&H0607C4EC)
DIBUTTON_FLYINGH_GLANCE_UP_LINK = 101172448 (&H0607C4E0)
DIBUTTON_FLYINGH_MENU = 100664573 (&H060004FD)
DIBUTTON_FLYINGH_PAUSE = 100680956 (&H060044FC)
DIBUTTON_FLYINGH_SLOWER_LINK = 100916456 (&H0603DCE8)
DIBUTTON_FLYINGH_TARGET = 100668419 (&H06001403)
DIBUTTON_FLYINGH_VIEW = 100688901 (&H06006405)
DIBUTTON_FLYINGH_WEAPONS = 100668418 (&H06001402)
DIBUTTON_FLYINGM_BRAKE_LINK = 84174048 (&H050464E0)
DIBUTTON_FLYINGM_COUNTER = 83909636 (&H05005C04)
DIBUTTON_FLYINGM_DEVICE = 83903742 (&H050044FE)
DIBUTTON_FLYINGM_DISPLAY = 83911686 (&H05006406)
DIBUTTON_FLYINGM_FASTER_LINK = 84137184 (&H0503D4E0)
DIBUTTON_FLYINGM_FIRE = 83889153 (&H05000C01)
DIBUTTON_FLYINGM_FIRESECONDARY = 83905546 (&H05004C09)
DIBUTTON_FLYINGM_FLAPSDOWN = 83907592 (&H05005408)
DIBUTTON_FLYINGM_FLAPSUP = 83907591 (&H05005407)
DIBUTTON_FLYINGM_GEAR = 83911689 (&H0500640A)
DIBUTTON_FLYINGM_GLANCE_DOWN_LINK = 84395240 (&H0507C4E8)
DIBUTTON_FLYINGM_GLANCE_LEFT_LINK = 84395236 (&H0507C4E4)
DIBUTTON_FLYINGM_GLANCE_RIGHT_LINK = 84395244 (&H0507C4EC)
DIBUTTON_FLYINGM_GLANCE_UP_LINK = 84395232 (&H0507C4E0)
DIBUTTON_FLYINGM_MENU = 83887357 (&H050004FD)
DIBUTTON_FLYINGM_PAUSE = 83903740 (&H050044FC)
DIBUTTON_FLYINGM_SLOWER_LINK = 84137192 (&H0503D4E8)
DIBUTTON_FLYINGM_TARGET = 83889155 (&H05000C03)
DIBUTTON_FLYINGM_VIEW = 83911685 (&H05006405)
DIBUTTON_FLYINGM_WEAPONS = 83889154 (&H05000C02)
DIBUTTON_FOOTBALLD_AUDIBLE = 385893387 (&H1700440B)
DIBUTTON_FOOTBALLD_BACK_LINK = 385959144 (&H170144E8)
DIBUTTON_FOOTBALLD_BULLRUSH = 385893385 (&H17004409)
DIBUTTON_FOOTBALLD_DEVICE = 385893630 (&H170044FE)
DIBUTTON_FOOTBALLD_FAKE = 385876997 (&H17000405)
DIBUTTON_FOOTBALLD_FORWARD_LINK = 385959136 (&H170144E0)
DIBUTTON_FOOTBALLD_JUMP = 385876995 (&H17000403)
DIBUTTON_FOOTBALLD_LEFT_LINK = 385926372 (&H1700C4E4)
DIBUTTON_FOOTBALLD_MENU = 385877245 (&H170004FD)
DIBUTTON_FOOTBALLD_PAUSE = 385893628 (&H170044FC)
DIBUTTON_FOOTBALLD_PLAY = 385876993 (&H17000401)

```
DIBUTTON_FOOTBALLD_RIGHT_LINK     = 385926380 (&H1700C4EC)
DIBUTTON_FOOTBALLD_RIP            = 385893386 (&H1700440A)
DIBUTTON_FOOTBALLD_SELECT         = 385876994 (&H17000402)
DIBUTTON_FOOTBALLD_SPIN           = 385893383 (&H17004407)
DIBUTTON_FOOTBALLD_SUBSTITUTE     = 385893389 (&H1700440D)
DIBUTTON_FOOTBALLD_SUPERTACKLE    = 385876998 (&H17000406)
DIBUTTON_FOOTBALLD_SWIM           = 385893384 (&H17004408)
DIBUTTON_FOOTBALLD_TACKLE         = 385876996 (&H17000404)
DIBUTTON_FOOTBALLD_ZOOM           = 385893388 (&H1700440C)
DIBUTTON_FOOTBALLO_BACK_LINK      = 369181928 (&H160144E8)
DIBUTTON_FOOTBALLO_DEVICE         = 369116414 (&H160044FE)
DIBUTTON_FOOTBALLO_DIVE           = 369116169 (&H16004409)
DIBUTTON_FOOTBALLO_FORWARD_LINK   = 369181920 (&H160144E0)
DIBUTTON_FOOTBALLO_JUKE           = 369116166 (&H16004406)
DIBUTTON_FOOTBALLO_JUMP           = 369099777 (&H16000401)
DIBUTTON_FOOTBALLO_LEFT_LINK      = 369149156 (&H1600C4E4)
DIBUTTON_FOOTBALLO_LEFTARM        = 369099778 (&H16000402)
DIBUTTON_FOOTBALLO_MENU           = 369100029 (&H160004FD)
DIBUTTON_FOOTBALLO_PAUSE          = 369116412 (&H160044FC)
DIBUTTON_FOOTBALLO_RIGHT_LINK     = 369149164 (&H1600C4EC)
DIBUTTON_FOOTBALLO_RIGHTARM       = 369099779 (&H16000403)
DIBUTTON_FOOTBALLO_SHOULDER       = 369116167 (&H16004407)
DIBUTTON_FOOTBALLO_SPIN           = 369099781 (&H16000405)
DIBUTTON_FOOTBALLO_SUBSTITUTE     = 369116171 (&H1600440B)
DIBUTTON_FOOTBALLO_THROW          = 369099780 (&H16000404)
DIBUTTON_FOOTBALLO_TURBO          = 369116168 (&H16004408)
DIBUTTON_FOOTBALLO_ZOOM           = 369116170 (&H1600440A)
DIBUTTON_FOOTBALLP_DEVICE         = 335561982 (&H140044FE)
DIBUTTON_FOOTBALLP_HELP           = 335545347 (&H14000403)
DIBUTTON_FOOTBALLP_MENU           = 335545597 (&H140004FD)
DIBUTTON_FOOTBALLP_PAUSE          = 335561980 (&H140044FC)
DIBUTTON_FOOTBALLP_PLAY           = 335545345 (&H14000401)
DIBUTTON_FOOTBALLP_SELECT         = 335545346 (&H14000402)
DIBUTTON_FOOTBALLQ_AUDIBLE        = 352338953 (&H15004409)
DIBUTTON_FOOTBALLQ_BACK_LINK      = 352404712 (&H150144E8)
DIBUTTON_FOOTBALLQ_DEVICE         = 352339198 (&H150044FE)
DIBUTTON_FOOTBALLQ_FAKE           = 352322566 (&H15000406)
DIBUTTON_FOOTBALLQ_FAKESNAP       = 352338951 (&H15004407)
DIBUTTON_FOOTBALLQ_FORWARD_LINK   = 352404704 (&H150144E0)
DIBUTTON_FOOTBALLQ_JUMP           = 352322563 (&H15000403)
DIBUTTON_FOOTBALLQ_LEFT_LINK      = 352371940 (&H1500C4E4)
DIBUTTON_FOOTBALLQ_MENU           = 352322813 (&H150004FD)
DIBUTTON_FOOTBALLQ_MOTION         = 352338952 (&H15004408)
DIBUTTON_FOOTBALLQ_PASS           = 352322565 (&H15000405)
DIBUTTON_FOOTBALLQ_PAUSE          = 352339196 (&H150044FC)
DIBUTTON_FOOTBALLQ_RIGHT_LINK     = 352371948 (&H1500C4EC)
```

DIBUTTON_FOOTBALLQ_SELECT = 352322561 (&H15000401)
DIBUTTON_FOOTBALLQ_SLIDE = 352322564 (&H15000404)
DIBUTTON_FOOTBALLQ_SNAP = 352322562 (&H15000402)
DIBUTTON_FPS_APPLY = 150995971 (&H09000403)
DIBUTTON_FPS_BACKWARD_LINK = 151078120 (&H090144E8)
DIBUTTON_FPS_CROUCH = 150995973 (&H09000405)
DIBUTTON_FPS_DEVICE = 151012606 (&H090044FE)
DIBUTTON_FPS_DISPLAY = 151012360 (&H09004408)
DIBUTTON_FPS_DODGE = 151012361 (&H09004409)
DIBUTTON_FPS_FIRE = 150995969 (&H09000401)
DIBUTTON_FPS_FIRESECONDARY = 151012364 (&H0900440C)
DIBUTTON_FPS_FORWARD_LINK = 151078112 (&H090144E0)
DIBUTTON_FPS_GLANCE_DOWN_LINK = 151110888 (&H0901C4E8)
DIBUTTON_FPS_GLANCE_UP_LINK = 151110880 (&H0901C4E0)
DIBUTTON_FPS_GLANCEL = 151012362 (&H0900440A)
DIBUTTON_FPS_GLANCER = 151012363 (&H0900440B)
DIBUTTON_FPS_JUMP = 150995974 (&H09000406)
DIBUTTON_FPS_MENU = 150996221 (&H090004FD)
DIBUTTON_FPS_PAUSE = 151012604 (&H090044FC)
DIBUTTON_FPS_ROTATE_LEFT_LINK = 151045348 (&H0900C4E4)
DIBUTTON_FPS_ROTATE_RIGHT_LINK = 151045356 (&H0900C4EC)
DIBUTTON_FPS_SELECT = 150995972 (&H09000404)
DIBUTTON_FPS_STRAFE = 150995975 (&H09000407)
DIBUTTON_FPS_WEAPONS = 150995970 (&H09000402)
DIBUTTON_GOLF_BACK_LINK = 402736360 (&H180144E8)
DIBUTTON_GOLF_DEVICE = 402670846 (&H180044FE)
DIBUTTON_GOLF_DOWN = 402654212 (&H18000404)
DIBUTTON_GOLF_FLYBY = 402654214 (&H18000406)
DIBUTTON_GOLF_FORWARD_LINK = 402736352 (&H180144E0)
DIBUTTON_GOLF_LEFT_LINK = 402703588 (&H1800C4E4)
DIBUTTON_GOLF_MENU = 402654461 (&H180004FD)
DIBUTTON_GOLF_PAUSE = 402670844 (&H180044FC)
DIBUTTON_GOLF_RIGHT_LINK = 402703596 (&H1800C4EC)
DIBUTTON_GOLF_SELECT = 402654210 (&H18000402)
DIBUTTON_GOLF_SUBSTITUTE = 402670601 (&H18004409)
DIBUTTON_GOLF_SWING = 402654209 (&H18000401)
DIBUTTON_GOLF_TERRAIN = 402654213 (&H18000405)
DIBUTTON_GOLF_TIMEOUT = 402670600 (&H18004408)
DIBUTTON_GOLF_UP = 402654211 (&H18000403)
DIBUTTON_GOLF_ZOOM = 402670599 (&H18004407)
DIBUTTON_HOCKEYD_BACK_LINK = 436290792 (&H1A0144E8)
DIBUTTON_HOCKEYD_BLOCK = 436208644 (&H1A000404)
DIBUTTON_HOCKEYD_BURST = 436208643 (&H1A000403)
DIBUTTON_HOCKEYD_DEVICE = 436225278 (&H1A0044FE)
DIBUTTON_HOCKEYD_FAKE = 436208645 (&H1A000405)
DIBUTTON_HOCKEYD_FORWARD_LINK = 436290784 (&H1A0144E0)

```
DIBUTTON_HOCKEYD_LEFT_LINK       = 436258020 (&H1A00C4E4)
DIBUTTON_HOCKEYD_MENU            = 436208893 (&H1A0004FD)
DIBUTTON_HOCKEYD_PAUSE           = 436225276 (&H1A0044FC)
DIBUTTON_HOCKEYD_PLAYER          = 436208641 (&H1A000401)
DIBUTTON_HOCKEYD_RIGHT_LINK      = 436258028 (&H1A00C4EC)
DIBUTTON_HOCKEYD_STEAL           = 436208642 (&H1A000402)
DIBUTTON_HOCKEYD_STRATEGY        = 436225031 (&H1A004407)
DIBUTTON_HOCKEYD_SUBSTITUTE      = 436225033 (&H1A004409)
DIBUTTON_HOCKEYD_TIMEOUT         = 436225032 (&H1A004408)
DIBUTTON_HOCKEYD_ZOOM            = 436225030 (&H1A004406)
DIBUTTON_HOCKEYG_BACK_LINK       = 453068008 (&H1B0144E8)
DIBUTTON_HOCKEYG_BLOCK           = 452985860 (&H1B000404)
DIBUTTON_HOCKEYG_DEVICE          = 453002494 (&H1B0044FE)
DIBUTTON_HOCKEYG_FORWARD_LINK    = 453068000 (&H1B0144E0)
DIBUTTON_HOCKEYG_LEFT_LINK       = 453035236 (&H1B00C4E4)
DIBUTTON_HOCKEYG_MENU            = 452986109 (&H1B0004FD)
DIBUTTON_HOCKEYG_PASS            = 452985857 (&H1B000401)
DIBUTTON_HOCKEYG_PAUSE           = 453002492 (&H1B0044FC)
DIBUTTON_HOCKEYG_POKE            = 452985858 (&H1B000402)
DIBUTTON_HOCKEYG_RIGHT_LINK      = 453035244 (&H1B00C4EC)
DIBUTTON_HOCKEYG_STEAL           = 452985859 (&H1B000403)
DIBUTTON_HOCKEYG_STRATEGY        = 453002246 (&H1B004406)
DIBUTTON_HOCKEYG_SUBSTITUTE      = 453002248 (&H1B004408)
DIBUTTON_HOCKEYG_TIMEOUT         = 453002247 (&H1B004407)
DIBUTTON_HOCKEYG_ZOOM            = 453002245 (&H1B004405)
DIBUTTON_HOCKEYO_BACK_LINK       = 419513576 (&H190144E8)
DIBUTTON_HOCKEYO_BURST           = 419431427 (&H19000403)
DIBUTTON_HOCKEYO_DEVICE          = 419448062 (&H190044FE)
DIBUTTON_HOCKEYO_FAKE            = 419431429 (&H19000405)
DIBUTTON_HOCKEYO_FORWARD_LINK    = 419513568 (&H190144E0)
DIBUTTON_HOCKEYO_LEFT_LINK       = 419480804 (&H1900C4E4)
DIBUTTON_HOCKEYO_MENU            = 419431677 (&H190004FD)
DIBUTTON_HOCKEYO_PASS            = 419431426 (&H19000402)
DIBUTTON_HOCKEYO_PAUSE           = 419448060 (&H190044FC)
DIBUTTON_HOCKEYO_RIGHT_LINK      = 419480812 (&H1900C4EC)
DIBUTTON_HOCKEYO_SHOOT           = 419431425 (&H19000401)
DIBUTTON_HOCKEYO_SPECIAL         = 419431428 (&H19000404)
DIBUTTON_HOCKEYO_STRATEGY        = 419447815 (&H19004407)
DIBUTTON_HOCKEYO_SUBSTITUTE      = 419447817 (&H19004409)
DIBUTTON_HOCKEYO_TIMEOUT         = 419447816 (&H19004408)
DIBUTTON_HOCKEYO_ZOOM            = 419447814 (&H19004406)
DIBUTTON_HUNTING_AIM             = 218104834 (&H0D000402)
DIBUTTON_HUNTING_BACK_LINK       = 218186984 (&H0D0144E8)
DIBUTTON_HUNTING_BINOCULAR       = 218104836 (&H0D000404)
DIBUTTON_HUNTING_CALL            = 218104837 (&H0D000405)
DIBUTTON_HUNTING_CROUCH          = 218121225 (&H0D004409)
```

DIBUTTON_HUNTING_DEVICE = 218121470 (&H0D0044FE)
DIBUTTON_HUNTING_DISPLAY = 218121224 (&H0D004408)
DIBUTTON_HUNTING_FIRE = 218104833 (&H0D000401)
DIBUTTON_HUNTING_FIRESECONDARY = 218121227 (&H0D00440B)
DIBUTTON_HUNTING_FORWARD_LINK = 218186976 (&H0D0144E0)
DIBUTTON_HUNTING_JUMP = 218121226 (&H0D00440A)
DIBUTTON_HUNTING_LEFT_LINK = 218154212 (&H0D00C4E4)
DIBUTTON_HUNTING_MAP = 218104838 (&H0D000406)
DIBUTTON_HUNTING_MENU = 218105085 (&H0D0004FD)
DIBUTTON_HUNTING_PAUSE = 218121468 (&H0D0044FC)
DIBUTTON_HUNTING_RIGHT_LINK = 218154220 (&H0D00C4EC)
DIBUTTON_HUNTING_ROTATE_LEFT_LINK = 218252516 (&H0D0244E4)
DIBUTTON_HUNTING_ROTATE_RIGHT_LINK= 218252524 (&H0D0244EC)
DIBUTTON_HUNTING_SPECIAL = 218104839 (&H0D000407)
DIBUTTON_HUNTING_WEAPON = 218104835 (&H0D000403)
DIBUTTON_MECHA_BACK_LINK = 687949032 (&H290144E8)
DIBUTTON_MECHA_CENTER = 687883271 (&H29004407)
DIBUTTON_MECHA_DEVICE = 687883518 (&H290044FE)
DIBUTTON_MECHA_FASTER_LINK = 688112864 (&H2903C4E0)
DIBUTTON_MECHA_FIRE = 687866881 (&H29000401)
DIBUTTON_MECHA_FIRESECONDARY = 687883273 (&H29004409)
DIBUTTON_MECHA_FORWARD_LINK = 687949024 (&H290144E0)
DIBUTTON_MECHA_JUMP = 687866886 (&H29000406)
DIBUTTON_MECHA_LEFT_LINK = 687916260 (&H2900C4E4)
DIBUTTON_MECHA_MENU = 687867133 (&H290004FD)
DIBUTTON_MECHA_PAUSE = 687883516 (&H290044FC)
DIBUTTON_MECHA_REVERSE = 687866884 (&H29000404)
DIBUTTON_MECHA_RIGHT_LINK = 687916268 (&H2900C4EC)
DIBUTTON_MECHA_ROTATE_LEFT_LINK = 688014564 (&H290244E4)
DIBUTTON_MECHA_ROTATE_RIGHT_LINK = 688014572 (&H290244EC)
DIBUTTON_MECHA_SLOWER_LINK = 688112872 (&H2903C4E8)
DIBUTTON_MECHA_TARGET = 687866883 (&H29000403)
DIBUTTON_MECHA_VIEW = 687883272 (&H29004408)
DIBUTTON_MECHA_WEAPONS = 687866882 (&H29000402)
DIBUTTON_MECHA_ZOOM = 687866885 (&H29000405)
DIBUTTON_RACQUET_BACK_LINK = 536954088 (&H200144E8)
DIBUTTON_RACQUET_BACKSWING = 536871938 (&H20000402)
DIBUTTON_RACQUET_DEVICE = 536888574 (&H200044FE)
DIBUTTON_RACQUET_FORWARD_LINK = 536954080 (&H200144E0)
DIBUTTON_RACQUET_LEFT_LINK = 536921316 (&H2000C4E4)
DIBUTTON_RACQUET_MENU = 536872189 (&H200004FD)
DIBUTTON_RACQUET_PAUSE = 536888572 (&H200044FC)
DIBUTTON_RACQUET_RIGHT_LINK = 536921324 (&H2000C4EC)
DIBUTTON_RACQUET_SELECT = 536871941 (&H20000405)
DIBUTTON_RACQUET_SMASH = 536871939 (&H20000403)
DIBUTTON_RACQUET_SPECIAL = 536871940 (&H20000404)

DIBUTTON_RACQUET_SUBSTITUTE = 536888327 (&H20004407)
DIBUTTON_RACQUET_SWING = 536871937 (&H20000401)
DIBUTTON_RACQUET_TIMEOUT = 536888326 (&H20004406)
DIBUTTON_REMOTE_ADJUST = 654334990 (&H27005C0E)
DIBUTTON_REMOTE_CABLE = 654334985 (&H27005C09)
DIBUTTON_REMOTE_CD = 654334986 (&H27005C0A)
DIBUTTON_REMOTE_CHANGE = 654320646 (&H27002406)
DIBUTTON_REMOTE_CUE = 654320644 (&H27002404)
DIBUTTON_REMOTE_DEVICE = 654329086 (&H270044FE)
DIBUTTON_REMOTE_DIGIT0 = 654332943 (&H2700540F)
DIBUTTON_REMOTE_DIGIT1 = 654332944 (&H27005410)
DIBUTTON_REMOTE_DIGIT2 = 654332945 (&H27005411)
DIBUTTON_REMOTE_DIGIT3 = 654332946 (&H27005412)
DIBUTTON_REMOTE_DIGIT4 = 654332947 (&H27005413)
DIBUTTON_REMOTE_DIGIT5 = 654332948 (&H27005414)
DIBUTTON_REMOTE_DIGIT6 = 654332949 (&H27005415)
DIBUTTON_REMOTE_DIGIT7 = 654332950 (&H27005416)
DIBUTTON_REMOTE_DIGIT8 = 654332951 (&H27005417)
DIBUTTON_REMOTE_DIGIT9 = 654332952 (&H27005418)
DIBUTTON_REMOTE_DVD = 654334989 (&H27005C0D)
DIBUTTON_REMOTE_MENU = 654312701 (&H270004FD)
DIBUTTON_REMOTE_MUTE = 654312449 (&H27000401)
DIBUTTON_REMOTE_PAUSE = 654329084 (&H270044FC)
DIBUTTON_REMOTE_PLAY = 654320643 (&H27002403)
DIBUTTON_REMOTE_RECORD = 654320647 (&H27002407)
DIBUTTON_REMOTE_REVIEW = 654320645 (&H27002405)
DIBUTTON_REMOTE_SELECT = 654312450 (&H27000402)
DIBUTTON_REMOTE_TUNER = 654334988 (&H27005C0C)
DIBUTTON_REMOTE_TV = 654334984 (&H27005C08)
DIBUTTON_REMOTE_VCR = 654334987 (&H27005C0B)
DIBUTTON_SKIING_CAMERA = 486540291 (&H1D000403)
DIBUTTON_SKIING_CROUCH = 486540290 (&H1D000402)
DIBUTTON_SKIING_DEVICE = 486556926 (&H1D0044FE)
DIBUTTON_SKIING_FASTER_LINK = 486622432 (&H1D0144E0)
DIBUTTON_SKIING_JUMP = 486540289 (&H1D000401)
DIBUTTON_SKIING_LEFT_LINK = 486589668 (&H1D00C4E4)
DIBUTTON_SKIING_MENU = 486540541 (&H1D0004FD)
DIBUTTON_SKIING_PAUSE = 486556924 (&H1D0044FC)
DIBUTTON_SKIING_RIGHT_LINK = 486589676 (&H1D00C4EC)
DIBUTTON_SKIING_SELECT = 486540293 (&H1D000405)
DIBUTTON_SKIING_SLOWER_LINK = 486622440 (&H1D0144E8)
DIBUTTON_SKIING_SPECIAL1 = 486540292 (&H1D000404)
DIBUTTON_SKIING_SPECIAL2 = 486540294 (&H1D000406)
DIBUTTON_SKIING_ZOOM = 486556679 (&H1D004407)
DIBUTTON_SOCCERD_BACK_LINK = 520176872 (&H1F0144E8)
DIBUTTON_SOCCERD_BLOCK = 520094721 (&H1F000401)

DIBUTTON_SOCCERD_CLEAR = 520111114 (&H1F00440A)
DIBUTTON_SOCCERD_DEVICE = 520111358 (&H1F0044FE)
DIBUTTON_SOCCERD_FAKE = 520094723 (&H1F000403)
DIBUTTON_SOCCERD_FORWARD_LINK = 520176864 (&H1F0144E0)
DIBUTTON_SOCCERD_FOUL = 520111112 (&H1F004408)
DIBUTTON_SOCCERD_GOALIECHARGE = 520111115 (&H1F00440B)
DIBUTTON_SOCCERD_HEAD = 520111113 (&H1F004409)
DIBUTTON_SOCCERD_LEFT_LINK = 520144100 (&H1F00C4E4)
DIBUTTON_SOCCERD_MENU = 520094973 (&H1F0004FD)
DIBUTTON_SOCCERD_PAUSE = 520111356 (&H1F0044FC)
DIBUTTON_SOCCERD_PLAYER = 520094724 (&H1F000404)
DIBUTTON_SOCCERD_RIGHT_LINK = 520144108 (&H1F00C4EC)
DIBUTTON_SOCCERD_SELECT = 520094726 (&H1F000406)
DIBUTTON_SOCCERD_SLIDE = 520094727 (&H1F000407)
DIBUTTON_SOCCERD_SPECIAL = 520094725 (&H1F000405)
DIBUTTON_SOCCERD_STEAL = 520094722 (&H1F000402)
DIBUTTON_SOCCERD_SUBSTITUTE = 520111116 (&H1F00440C)
DIBUTTON_SOCCERO_BACK_LINK = 503399656 (&H1E0144E8)
DIBUTTON_SOCCERO_CONTROL = 503333900 (&H1E00440C)
DIBUTTON_SOCCERO_DEVICE = 503334142 (&H1E0044FE)
DIBUTTON_SOCCERO_FAKE = 503317507 (&H1E000403)
DIBUTTON_SOCCERO_FORWARD_LINK = 503399648 (&H1E0144E0)
DIBUTTON_SOCCERO_HEAD = 503333901 (&H1E00440D)
DIBUTTON_SOCCERO_LEFT_LINK = 503366884 (&H1E00C4E4)
DIBUTTON_SOCCERO_MENU = 503317757 (&H1E0004FD)
DIBUTTON_SOCCERO_PASS = 503317506 (&H1E000402)
DIBUTTON_SOCCERO_PASSTHRU = 503333898 (&H1E00440A)
DIBUTTON_SOCCERO_PAUSE = 503334140 (&H1E0044FC)
DIBUTTON_SOCCERO_PLAYER = 503317508 (&H1E000404)
DIBUTTON_SOCCERO_RIGHT_LINK = 503366892 (&H1E00C4EC)
DIBUTTON_SOCCERO_SELECT = 503317510 (&H1E000406)
DIBUTTON_SOCCERO_SHOOT = 503317505 (&H1E000401)
DIBUTTON_SOCCERO_SHOOTHIGH = 503333897 (&H1E004409)
DIBUTTON_SOCCERO_SHOOTLOW = 503333896 (&H1E004408)
DIBUTTON_SOCCERO_SPECIAL1 = 503317509 (&H1E000405)
DIBUTTON_SOCCERO_SPRINT = 503333899 (&H1E00440B)
DIBUTTON_SOCCERO_SUBSTITUTE = 503333895 (&H1E004407)
DIBUTTON_SPACESIM_BACKWARD_LINK = 117523688 (&H070144E8)
DIBUTTON_SPACESIM_DEVICE = 117458174 (&H070044FE)
DIBUTTON_SPACESIM_DISPLAY = 117457925 (&H07004405)
DIBUTTON_SPACESIM_FASTER_LINK = 117687520 (&H0703C4E0)
DIBUTTON_SPACESIM_FIRE = 117441537 (&H07000401)
DIBUTTON_SPACESIM_FIRESECONDARY = 117457929 (&H07004409)
DIBUTTON_SPACESIM_FORWARD_LINK = 117523680 (&H070144E0)
DIBUTTON_SPACESIM_GEAR = 117457928 (&H07004408)

DIBUTTON_SPACESIM_GLANCE_DOWN_LINK= 117949672 (&H0707C4E8)
DIBUTTON_SPACESIM_GLANCE_LEFT_LINK= 117949668 (&H0707C4E4)
DIBUTTON_SPACESIM_GLANCE_RIGHT_LINK=117949676 (&H0707C4EC)
DIBUTTON_SPACESIM_GLANCE_UP_LINK = 117949664 (&H0707C4E0)
DIBUTTON_SPACESIM_LEFT_LINK = 117490916 (&H0700C4E4)
DIBUTTON_SPACESIM_LOWER = 117457927 (&H07004407)
DIBUTTON_SPACESIM_MENU = 117441789 (&H070004FD)
DIBUTTON_SPACESIM_PAUSE = 117458172 (&H070044FC)
DIBUTTON_SPACESIM_RAISE = 117457926 (&H07004406)
DIBUTTON_SPACESIM_RIGHT_LINK = 117490924 (&H0700C4EC)
DIBUTTON_SPACESIM_SLOWER_LINK = 117687528 (&H0703C4E8)
DIBUTTON_SPACESIM_TARGET = 117441539 (&H07000403)
DIBUTTON_SPACESIM_TURN_LEFT_LINK = 117589220 (&H070244E4)
DIBUTTON_SPACESIM_TURN_RIGHT_LINK = 117589228 (&H070244EC)
DIBUTTON_SPACESIM_VIEW = 117457924 (&H07004404)
DIBUTTON_SPACESIM_WEAPONS = 117441538 (&H07000402)
DIBUTTON_STRATEGYR_APPLY = 184550402 (&H0B000402)
DIBUTTON_STRATEGYR_ATTACK = 184550404 (&H0B000404)
DIBUTTON_STRATEGYR_BACK_LINK = 184632552 (&H0B0144E8)
DIBUTTON_STRATEGYR_CAST = 184550405 (&H0B000405)
DIBUTTON_STRATEGYR_CROUCH = 184550406 (&H0B000406)
DIBUTTON_STRATEGYR_DEVICE = 184567038 (&H0B0044FE)
DIBUTTON_STRATEGYR_DISPLAY = 184566793 (&H0B004409)
DIBUTTON_STRATEGYR_FORWARD_LINK = 184632544 (&H0B0144E0)
DIBUTTON_STRATEGYR_GET = 184550401 (&H0B000401)
DIBUTTON_STRATEGYR_JUMP = 184550407 (&H0B000407)
DIBUTTON_STRATEGYR_LEFT_LINK = 184599780 (&H0B00C4E4)
DIBUTTON_STRATEGYR_MAP = 184566792 (&H0B004408)
DIBUTTON_STRATEGYR_MENU = 184550653 (&H0B0004FD)
DIBUTTON_STRATEGYR_PAUSE = 184567036 (&H0B0044FC)
DIBUTTON_STRATEGYR_RIGHT_LINK = 184599788 (&H0B00C4EC)
DIBUTTON_STRATEGYR_ROTATE_LEFT_LINK=184698084 (&H0B0244E4)
DIBUTTON_STRATEGYR_ROTATE_RIGHT_LINK=184698092 (&HB0244EC)
DIBUTTON_STRATEGYR_SELECT = 184550403 (&H0B000403)
DIBUTTON_STRATEGYT_APPLY = 201327619 (&H0C000403)
DIBUTTON_STRATEGYT_BACK_LINK = 201409768 (&H0C0144E8)
DIBUTTON_STRATEGYT_DEVICE = 201344254 (&H0C0044FE)
DIBUTTON_STRATEGYT_DISPLAY = 201344008 (&H0C004408)
DIBUTTON_STRATEGYT_FORWARD_LINK = 201409760 (&H0C0144E0)
DIBUTTON_STRATEGYT_INSTRUCT = 201327618 (&H0C000402)
DIBUTTON_STRATEGYT_LEFT_LINK = 201376996 (&H0C00C4E4)
DIBUTTON_STRATEGYT_MAP = 201344007 (&H0C004407)

```
DIBUTTON_STRATEGYT_MENU          = 201327869 (&H0C0004FD)
DIBUTTON_STRATEGYT_PAUSE         = 201344252 (&H0C0044FC)
DIBUTTON_STRATEGYT_RIGHT_LINK    = 201377004 (&H0C00C4EC)
DIBUTTON_STRATEGYT_SELECT        = 201327617 (&H0C000401)
DIBUTTON_STRATEGYT_TEAM          = 201327620 (&H0C000404)
DIBUTTON_STRATEGYT_TURN          = 201327621 (&H0C000405)
DIBUTTON_STRATEGYT_ZOOM          = 201344006 (&H0C004406)
DIBUTTON_TPS_ACTION              = 167773186 (&H0A000402)
DIBUTTON_TPS_BACKWARD_LINK       = 167855336 (&H0A0144E8)
DIBUTTON_TPS_DEVICE              = 167789822 (&H0A0044FE)
DIBUTTON_TPS_DODGE               = 167789577 (&H0A004409)
DIBUTTON_TPS_FORWARD_LINK        = 167855328 (&H0A0144E0)
DIBUTTON_TPS_GLANCE_DOWN_LINK    = 168281320 (&H0A07C4E8)
DIBUTTON_TPS_GLANCE_LEFT_LINK    = 168281316 (&H0A07C4E4)
DIBUTTON_TPS_GLANCE_RIGHT_LINK   = 168281324 (&H0A07C4EC)
DIBUTTON_TPS_GLANCE_UP_LINK      = 168281312 (&H0A07C4E0)
DIBUTTON_TPS_INVENTORY           = 167789578 (&H0A00440A)
DIBUTTON_TPS_JUMP                = 167773189 (&H0A000405)
DIBUTTON_TPS_MENU                = 167773437 (&H0A0004FD)
DIBUTTON_TPS_PAUSE               = 167789820 (&H0A0044FC)
DIBUTTON_TPS_RUN                 = 167773185 (&H0A000401)
DIBUTTON_TPS_SELECT              = 167773187 (&H0A000403)
DIBUTTON_TPS_STEPLEFT            = 167789575 (&H0A004407)
DIBUTTON_TPS_STEPRIGHT           = 167789576 (&H0A004408)
DIBUTTON_TPS_TURN_LEFT_LINK      = 167920868 (&H0A0244E4)
DIBUTTON_TPS_TURN_RIGHT_LINK     = 167920876 (&H0A0244EC)
DIBUTTON_TPS_USE                 = 167773188 (&H0A000404)
DIBUTTON_TPS_VIEW                = 167789574 (&H0A004406)
DIHATSWITCH_2DCONTROL_HATSWITCH  = 587220481 (&H23004601)
DIHATSWITCH_3DCONTROL_HATSWITCH  = 603997697 (&H24004601)
DIHATSWITCH_ARCADEP_VIEW         = 570443265 (&H22004601)
DIHATSWITCH_ARCADES_VIEW         = 553666049 (&H21004601)
DIHATSWITCH_BBALLD_GLANCE        = 318785025 (&H13004601)
DIHATSWITCH_BBALLO_GLANCE        = 302007809 (&H12004601)
DIHATSWITCH_BIKINGM_SCROLL       = 469779969 (&H1C004601)
DIHATSWITCH_CADF_HATSWITCH       = 620774913 (&H25004601)
DIHATSWITCH_CADM_HATSWITCH       = 637552129 (&H26004601)
DIHATSWITCH_DRIVINGC_GLANCE      =  33572353 (&H02004601)
DIHATSWITCH_DRIVINGR_GLANCE      =  16795137 (&H01004601)
DIHATSWITCH_DRIVINGT_GLANCE      =  50349569 (&H03004601)
DIHATSWITCH_FIGHTINGH_SLIDE      = 134235649 (&H08004601)
DIHATSWITCH_FISHING_GLANCE       = 234898945 (&H0E004601)
DIHATSWITCH_FLYINGC_GLANCE       =  67126785 (&H04004601)
DIHATSWITCH_FLYINGH_GLANCE       = 100681217 (&H06004601)
DIHATSWITCH_FLYINGM_GLANCE       =  83904001 (&H05004601)
DIHATSWITCH_FPS_GLANCE           = 151012865 (&H09004601)
```

```
  DIHATSWITCH_GOLF_SCROLL          = 402671105 (&H18004601)
  DIHATSWITCH_HOCKEYD_SCROLL       = 436225537 (&H1A004601)
  DIHATSWITCH_HOCKEYG_SCROLL       = 453002753 (&H1B004601)
  DIHATSWITCH_HOCKEYO_SCROLL       = 419448321 (&H19004601)
  DIHATSWITCH_HUNTING_GLANCE       = 218121729 (&H0D004601)
  DIHATSWITCH_MECHA_GLANCE         = 687883777 (&H29004601)
  DIHATSWITCH_RACQUET_GLANCE       = 536888833 (&H20004601)
  DIHATSWITCH_SKIING_GLANCE        = 486557185 (&H1D004601)
  DIHATSWITCH_SOCCERD_GLANCE       = 520111617 (&H1F004601)
  DIHATSWITCH_SOCCERO_GLANCE       = 503334401 (&H1E004601)
  DIHATSWITCH_SPACESIM_GLANCE      = 117458433 (&H07004601)
  DIHATSWITCH_STRATEGYR_GLANCE     = 184567297 (&H0B004601)
  DIHATSWITCH_TPS_GLANCE           = 167790081 (&H0A004601)
  DIVIRTUAL_ARCADE_PLATFORM        = 570425344 (&H22000000)
  DIVIRTUAL_ARCADE_SIDE2SIDE       = 553648128 (&H21000000)
  DIVIRTUAL_BROWSER_CONTROL        = 671088640 (&H28000000)
  DIVIRTUAL_CAD_2DCONTROL          = 587202560 (&H23000000)
  DIVIRTUAL_CAD_3DCONTROL          = 603979776 (&H24000000)
  DIVIRTUAL_CAD_FLYBY              = 620756992 (&H25000000)
  DIVIRTUAL_CAD_MODEL              = 637534208 (&H26000000)
  DIVIRTUAL_DRIVING_COMBAT         =  33554432 (&H02000000)
  DIVIRTUAL_DRIVING_MECHA          = 687865856 (&H29000000)
  DIVIRTUAL_DRIVING_RACE           =  16777216 (&H01000000)
  DIVIRTUAL_DRIVING_TANK           =  50331648 (&H03000000)
  DIVIRTUAL_FIGHTING_FPS           = 150994944 (&H09000000)
  DIVIRTUAL_FIGHTING_HAND2HAND     = 134217728 (&H08000000)
  DIVIRTUAL_FIGHTING_THIRDPERSON   = 167772160 (&H0A000000)
  DIVIRTUAL_FLYING_CIVILIAN        =  67108864 (&H04000000)
  DIVIRTUAL_FLYING_HELICOPTER      = 100663296 (&H06000000)
  DIVIRTUAL_FLYING_MILITARY        =  83886080 (&H05000000)
  DIVIRTUAL_REMOTE_CONTROL         = 654311424 (&H27000000)
  DIVIRTUAL_SPACESIM               = 117440512 (&H07000000)
  DIVIRTUAL_SPORTS_BASEBALL_BAT    = 251658240 (&H0F000000)
  DIVIRTUAL_SPORTS_BASEBALL_FIELD  = 285212672 (&H11000000)
  DIVIRTUAL_SPORTS_BASEBALL_PITCH  = 268435456 (&H10000000)
  DIVIRTUAL_SPORTS_BASKETBALL_DEFENSE=318767104
(&H13000000)
  DIVIRTUAL_SPORTS_BASKETBALL_OFFENSE=301989888
(&H12000000)
  DIVIRTUAL_SPORTS_BIKING_MOUNTAIN  = 469762048 (&H1C000000)
  DIVIRTUAL_SPORTS_FISHING         = 234881024 (&H0E000000)
  DIVIRTUAL_SPORTS_FOOTBALL_DEFENSE = 385875968 (&H17000000)
  DIVIRTUAL_SPORTS_FOOTBALL_FIELD  = 335544320 (&H14000000)
  DIVIRTUAL_SPORTS_FOOTBALL_OFFENSE = 369098752 (&H16000000)
  DIVIRTUAL_SPORTS_FOOTBALL_QBCK   = 352321536 (&H15000000)
  DIVIRTUAL_SPORTS_GOLF            = 402653184 (&H18000000)
```

```
  DIVIRTUAL_SPORTS_HOCKEY_DEFENSE   = 436207616 (&H1A000000)
  DIVIRTUAL_SPORTS_HOCKEY_GOALIE    = 452984832 (&H1B000000)
  DIVIRTUAL_SPORTS_HOCKEY_OFFENSE   = 419430400 (&H19000000)
  DIVIRTUAL_SPORTS_HUNTING          = 218103808 (&H0D000000)
  DIVIRTUAL_SPORTS_RACQUET          = 536870912 (&H20000000)
  DIVIRTUAL_SPORTS_SKIING           = 486539264 (&H1D000000)
  DIVIRTUAL_SPORTS_SOCCER_DEFENSE   = 520093696 (&H1F000000)
  DIVIRTUAL_SPORTS_SOCCER_OFFENSE   = 503316480 (&H1E000000)
  DIVIRTUAL_STRATEGY_ROLEPLAYING    = 184549376 (&H0B000000)
  DIVIRTUAL_STRATEGY_TURN           = 201326592 (&H0C000000)
End Enum
```

CONST_DIGFFSFLAGS

#Members of this enumeration describe the state of a force-feedback device. They are returned by the **DirectInputDevice8.GetForceFeedbackState** method.

```
Enum CONST_DIGFFSFLAGS
   DIGFFS_ACTUATORSOFF    =         32 (&H20)
   DIGFFS_ACTUATORSON     =         16 (&H10)
   DIGFFS_DEVICELOST      = -2147483648 (&H80000000)
   DIGFFS_EMPTY           =          1
   DIGFFS_PAUSED          =          4
   DIGFFS_POWEROFF        =        128 (&H80)
   DIGFFS_POWERON         =         64 (&H40)
   DIGFFS_SAFETYSWITCHOFF =        512 (&H200)
   DIGFFS_SAFETYSWITCHON  =        256 (&H100)
   DIGFFS_STOPPED         =          2
   DIGFFS_USERFFSWITCHOFF =       2048 (&H800)
   DIGFFS_USERFFSWITCHON  =       1024 (&H400)
End Enum
```

Constants

DIGFFS_ACTUATORSOFF
: The device's force-feedback actuators are disabled.

DIGFFS_ACTUATORSON
: The device's force-feedback actuators are enabled.

DIGFFS_DEVICELOST
: The device suffered an unexpected failure and is in an indeterminate state. It must be reset either by unacquiring and reacquiring the device, or by sending a DISFFC_RESET command.

DIGFFS_EMPTY
: The device has no downloaded effects.

IDH_CONST_DIGFFSFLAGS_dinput_vb

DIGFFS_PAUSED
Playback of all active effects has been paused.

DIGFFS_POWEROFF
The force-feedback system is not currently available. If the device cannot report the power state, then neither DIGFFS_POWERON nor DIGFFS_POWEROFF will be returned.

DIGFFS_POWERON
Power to the force-feedback system is currently available. If the device cannot report the power state, then neither DIGFFS_POWERON nor DIGFFS_POWEROFF will be returned.

DIGFFS_SAFETYSWITCHOFF
The safety switch is currently off, meaning that the device cannot operate. If the device cannot report the state of the safety switch, then neither DIGFFS_SAFETYSWITCHON nor DIGFFS_SAFETYSWITCHOFF will be returned.

DIGFFS_SAFETYSWITCHON
The safety switch is currently on, meaning that the device can operate. If the device cannot report the state of the safety switch, then neither DIGFFS_SAFETYSWITCHON nor DIGFFS_SAFETYSWITCHOFF will be returned.

DIGFFS_STOPPED
No effects are playing and the device is not paused.

DIGFFS_USERFFSWITCHOFF
The user force-feedback switch is currently off, meaning that the device cannot operate. If the device cannot report the state of the user force-feedback switch, then neither DIGFFS_USERFFSWITCHON nor DIGFFS_USERFFSWITCHOFF will be returned.

DIGFFS_USERFFSWITCHON
The user force-feedback switch is currently on, meaning that the device can operate. If the device cannot report the state of the user force-feedback switch, then neither DIGFFS_USERFFSWITCHON nor DIGFFS_USERFFSWITCHOFF will be returned.

CONST_DIJOYSTICKOFS

#Used to identify device objects in joystick data. The values of the constants are the offset of the data within the data format for immediate data. The same identifiers are used in buffered data packets.

```
Enum CONST_DIJOYSTICKOFS
   DIJOFS_BUTTON0  = 48 (&H30)
   DIJOFS_BUTTON1  = 49 (&H31)
   DIJOFS_BUTTON2  = 50 (&H32)
```

IDH_CONST_DIJOYSTICKOFS_dinput_vb

```
    DIJOFS_BUTTON3  = 51 (&H33)
    DIJOFS_BUTTON4  = 52 (&H34)
    DIJOFS_BUTTON5  = 53 (&H35)
    DIJOFS_BUTTON6  = 54 (&H36)
    DIJOFS_BUTTON7  = 55 (&H37)
    DIJOFS_BUTTON8  = 56 (&H38)
    DIJOFS_BUTTON9  = 57 (&H39)
    DIJOFS_BUTTON10 = 58 (&H3A)
    DIJOFS_BUTTON11 = 59 (&H3B)
    DIJOFS_BUTTON12 = 60 (&H3C)
    DIJOFS_BUTTON13 = 61 (&H3D)
    DIJOFS_BUTTON14 = 62 (&H3E)
    DIJOFS_BUTTON15 = 63 (&H3F)
    DIJOFS_BUTTON16 = 64 (&H40)
    DIJOFS_BUTTON17 = 65 (&H41)
    DIJOFS_BUTTON18 = 66 (&H42)
    DIJOFS_BUTTON19 = 67 (&H43)
    DIJOFS_BUTTON20 = 68 (&H44)
    DIJOFS_BUTTON21 = 69 (&H45)
    DIJOFS_BUTTON22 = 70 (&H46)
    DIJOFS_BUTTON23 = 71 (&H47)
    DIJOFS_BUTTON24 = 72 (&H48)
    DIJOFS_BUTTON25 = 73 (&H49)
    DIJOFS_BUTTON26 = 74 (&H4A)
    DIJOFS_BUTTON27 = 75 (&H4B)
    DIJOFS_BUTTON28 = 76 (&H4C)
    DIJOFS_BUTTON29 = 77 (&H4D)
    DIJOFS_BUTTON30 = 78 (&H4E)
    DIJOFS_BUTTON31 = 79 (&H4F)
    DIJOFS_POV0     = 32 (&H20)
    DIJOFS_POV1     = 36 (&H24)
    DIJOFS_POV2     = 40 (&H28)
    DIJOFS_POV3     = 44 (&H2C)
    DIJOFS_RX       = 12
    DIJOFS_RY       = 16 (&H10)
    DIJOFS_RZ       = 20 (&H14)
    DIJOFS_SLIDER0  = 24 (&H18)
    DIJOFS_SLIDER1  = 28 (&H1C)
    DIJOFS_X        =  0
    DIJOFS_Y        =  4
    DIJOFS_Z        =  8
End Enum
```

Constants

DIJOFS_BUTTON(*n*)

Offset of the data for button *n*.

DIJOFS_POV0, DIJOFS_POV1, DIJOFS_POV2, DIJOFS_POV3
Offset of the data for the point-of-view controller.

DIJOFS_RX, DIJOFS_RY, DIJOFS_RZ
Offset of the data for the axis rotation.

DIJOFS_SLIDER0, DIJOFS_SLIDER1
Offset of the data for the slider.

DIJOFS_X, DIJOFS_Y, DIJOFS_Z
Offset of the data for the axis.

CONST_DIKEYBOARDFLAGS

[#]Used to specify keys in the **lSemantic** member of the **DIACTION** type.

```
Enum CONST_DIKEYBOARDFLAGS
    DIKEYBOARD_0           = -2130705397 (&H8100040B)
    DIKEYBOARD_1           = -2130705406 (&H81000402)
    DIKEYBOARD_2           = -2130705405 (&H81000403)
    DIKEYBOARD_3           = -2130705404 (&H81000404)
    DIKEYBOARD_4           = -2130705403 (&H81000405)
    DIKEYBOARD_5           = -2130705402 (&H81000406)
    DIKEYBOARD_6           = -2130705401 (&H81000407)
    DIKEYBOARD_7           = -2130705400 (&H81000408)
    DIKEYBOARD_8           = -2130705399 (&H81000409)
    DIKEYBOARD_9           = -2130705398 (&H8100040A)
    DIKEYBOARD_A           = -2130705378 (&H8100041E)
    DIKEYBOARD_ABNT_C1     = -2130705293 (&H81000473)
    DIKEYBOARD_ABNT_C2     = -2130705282 (&H8100047E)
    DIKEYBOARD_ADD         = -2130705330 (&H8100044E)
    DIKEYBOARD_APOSTROPHE  = -2130705368 (&H81000428)
    DIKEYBOARD_APPS        = -2130705187 (&H810004DD)
    DIKEYBOARD_AT          = -2130705263 (&H81000491)
    DIKEYBOARD_AX          = -2130705258 (&H81000496)
    DIKEYBOARD_B           = -2130705360 (&H81000430)
    DIKEYBOARD_BACK        = -2130705394 (&H8100040E)
    DIKEYBOARD_BACKSLASH   = -2130705365 (&H8100042B)
    DIKEYBOARD_C           = -2130705362 (&H8100042E)
    DIKEYBOARD_CALCULATOR  = -2130705247 (&H810004A1)
    DIKEYBOARD_CAPITAL     = -2130705350 (&H8100043A)
    DIKEYBOARD_COLON       = -2130705262 (&H81000492)
    DIKEYBOARD_COMMA       = -2130705357 (&H81000433)
    DIKEYBOARD_CONVERT     = -2130705287 (&H81000479)
    DIKEYBOARD_D           = -2130705376 (&H81000420)
```

[#] IDH_CONST_DIKEYBOARDFLAGS_dinput_vb

```
DIKEYBOARD_DECIMAL      = -2130705325 (&H81000453)
DIKEYBOARD_DELETE       = -2130705197 (&H810004D3)
DIKEYBOARD_DIVIDE       = -2130705227 (&H810004B5)
DIKEYBOARD_DOWN         = -2130705200 (&H810004D0)
DIKEYBOARD_E            = -2130705390 (&H81000412)
DIKEYBOARD_END          = -2130705201 (&H810004CF)
DIKEYBOARD_EQUALS       = -2130705395 (&H8100040D)
DIKEYBOARD_ESCAPE       = -2130705407 (&H81000401)
DIKEYBOARD_F            = -2130705375 (&H81000421)
DIKEYBOARD_F1           = -2130705349 (&H8100043B)
DIKEYBOARD_F2           = -2130705348 (&H8100043C)
DIKEYBOARD_F3           = -2130705347 (&H8100043D)
DIKEYBOARD_F4           = -2130705346 (&H8100043E)
DIKEYBOARD_F5           = -2130705345 (&H8100043F)
DIKEYBOARD_F6           = -2130705344 (&H81000440)
DIKEYBOARD_F7           = -2130705343 (&H81000441)
DIKEYBOARD_F8           = -2130705342 (&H81000442)
DIKEYBOARD_F9           = -2130705341 (&H81000443)
DIKEYBOARD_F10          = -2130705340 (&H81000444)
DIKEYBOARD_F11          = -2130705321 (&H81000457)
DIKEYBOARD_F12          = -2130705320 (&H81000458)
DIKEYBOARD_F13          = -2130705308 (&H81000464)
DIKEYBOARD_F14          = -2130705307 (&H81000465)
DIKEYBOARD_F15          = -2130705306 (&H81000466)
DIKEYBOARD_G            = -2130705374 (&H81000422)
DIKEYBOARD_GRAVE        = -2130705367 (&H81000429)
DIKEYBOARD_H            = -2130705373 (&H81000423)
DIKEYBOARD_HOME         = -2130705209 (&H810004C7)
DIKEYBOARD_I            = -2130705385 (&H81000417)
DIKEYBOARD_INSERT       = -2130705198 (&H810004D2)
DIKEYBOARD_J            = -2130705372 (&H81000424)
DIKEYBOARD_K            = -2130705371 (&H81000425)
DIKEYBOARD_KANA         = -2130705296 (&H81000470)
DIKEYBOARD_KANJI        = -2130705260 (&H81000494)
DIKEYBOARD_L            = -2130705370 (&H81000426)
DIKEYBOARD_LBRACKET     = -2130705382 (&H8100041A)
DIKEYBOARD_LCONTROL     = -2130705379 (&H8100041D)
DIKEYBOARD_LEFT         = -2130705205 (&H810004CB)
DIKEYBOARD_LMENU        = -2130705352 (&H81000438)
DIKEYBOARD_LSHIFT       = -2130705366 (&H8100042A)
DIKEYBOARD_LWIN         = -2130705189 (&H810004DB)
DIKEYBOARD_M            = -2130705358 (&H81000432)
DIKEYBOARD_MAIL         = -2130705172 (&H810004EC)
DIKEYBOARD_MEDIASELECT  = -2130705171 (&H810004ED)
DIKEYBOARD_MEDIASTOP    = -2130705244 (&H810004A4)
DIKEYBOARD_MINUS        = -2130705396 (&H8100040C)
```

```
DIKEYBOARD_MULTIPLY     = -2130705353 (&H81000437)
DIKEYBOARD_MUTE         = -2130705248 (&H810004A0)
DIKEYBOARD_MYCOMPUTER   = -2130705173 (&H810004EB)
DIKEYBOARD_N            = -2130705359 (&H81000431)
DIKEYBOARD_NEXT         = -2130705199 (&H810004D1)
DIKEYBOARD_NEXTTRACK    = -2130705255 (&H81000499)
DIKEYBOARD_NOCONVERT    = -2130705285 (&H8100047B)
DIKEYBOARD_NUMLOCK      = -2130705339 (&H81000445)
DIKEYBOARD_NUMPAD0      = -2130705326 (&H81000452)
DIKEYBOARD_NUMPAD1      = -2130705329 (&H8100044F)
DIKEYBOARD_NUMPAD2      = -2130705328 (&H81000450)
DIKEYBOARD_NUMPAD3      = -2130705327 (&H81000451)
DIKEYBOARD_NUMPAD4      = -2130705333 (&H8100044B)
DIKEYBOARD_NUMPAD5      = -2130705332 (&H8100044C)
DIKEYBOARD_NUMPAD6      = -2130705331 (&H8100044D)
DIKEYBOARD_NUMPAD7      = -2130705337 (&H81000447)
DIKEYBOARD_NUMPAD8      = -2130705336 (&H81000448)
DIKEYBOARD_NUMPAD9      = -2130705335 (&H81000449)
DIKEYBOARD_NUMPADCOMMA  = -2130705229 (&H810004B3)
DIKEYBOARD_NUMPADENTER  = -2130705252 (&H8100049C)
DIKEYBOARD_NUMPADEQUALS = -2130705267 (&H8100048D)
DIKEYBOARD_O            = -2130705384 (&H81000418)
DIKEYBOARD_OEM_102      = -2130705322 (&H81000456)
DIKEYBOARD_P            = -2130705383 (&H81000419)
DIKEYBOARD_PAUSE        = -2130705211 (&H810004C5)
DIKEYBOARD_PERIOD       = -2130705356 (&H81000434)
DIKEYBOARD_PLAYPAUSE    = -2130705246 (&H810004A2)
DIKEYBOARD_POWER        = -2130705186 (&H810004DE)
DIKEYBOARD_PREVTRACK    = -2130705264 (&H81000490)
DIKEYBOARD_PRIOR        = -2130705207 (&H810004C9)
DIKEYBOARD_Q            = -2130705392 (&H81000410)
DIKEYBOARD_R            = -2130705389 (&H81000413)
DIKEYBOARD_RBRACKET     = -2130705381 (&H8100041B)
DIKEYBOARD_RCONTROL     = -2130705251 (&H8100049D)
DIKEYBOARD_RETURN       = -2130705380 (&H8100041C)
DIKEYBOARD_RIGHT        = -2130705203 (&H810004CD)
DIKEYBOARD_RMENU        = -2130705224 (&H810004B8)
DIKEYBOARD_RSHIFT       = -2130705354 (&H81000436)
DIKEYBOARD_RWIN         = -2130705188 (&H810004DC)
DIKEYBOARD_S            = -2130705377 (&H8100041F)
DIKEYBOARD_SCROLL       = -2130705338 (&H81000446)
DIKEYBOARD_SEMICOLON    = -2130705369 (&H81000427)
DIKEYBOARD_SLASH        = -2130705355 (&H81000435)
DIKEYBOARD_SLEEP        = -2130705185 (&H810004DF)
DIKEYBOARD_SPACE        = -2130705351 (&H81000439)
DIKEYBOARD_STOP         = -2130705259 (&H81000495)
```

```
    DIKEYBOARD_SUBTRACT     = -2130705334 (&H8100044A)
    DIKEYBOARD_SYSRQ        = -2130705225 (&H810004B7)
    DIKEYBOARD_T            = -2130705388 (&H81000414)
    DIKEYBOARD_TAB          = -2130705393 (&H8100040F)
    DIKEYBOARD_U            = -2130705386 (&H81000416)
    DIKEYBOARD_UNDERLINE    = -2130705261 (&H81000493)
    DIKEYBOARD_UNLABELED    = -2130705257 (&H81000497)
    DIKEYBOARD_UP           = -2130705208 (&H810004C8)
    DIKEYBOARD_V            = -2130705361 (&H8100042F)
    DIKEYBOARD_VOLUMEDOWN   = -2130705234 (&H810004AE)
    DIKEYBOARD_VOLUMEUP     = -2130705232 (&H810004B0)
    DIKEYBOARD_W            = -2130705391 (&H81000411)
    DIKEYBOARD_WAKE         = -2130705181 (&H810004E3)
    DIKEYBOARD_WEBBACK      = -2130705174 (&H810004EA)
    DIKEYBOARD_WEBFAVORITES = -2130705178 (&H810004E6)
    DIKEYBOARD_WEBFORWARD   = -2130705175 (&H810004E9)
    DIKEYBOARD_WEBHOME      = -2130705230 (&H810004B2)
    DIKEYBOARD_WEBREFRESH   = -2130705177 (&H810004E7)
    DIKEYBOARD_WEBSEARCH    = -2130705179 (&H810004E5)
    DIKEYBOARD_WEBSTOP      = -2130705176 (&H810004E8)
    DIKEYBOARD_X            = -2130705363 (&H8100042D)
    DIKEYBOARD_Y            = -2130705387 (&H81000415)
    DIKEYBOARD_YEN          = -2130705283 (&H8100047D)
    DIKEYBOARD_Z            = -2130705364 (&H8100042C)
End Enum
```

Constants

DIKEYBOARD_(*n*)
: Number keys 0 through 9 on the main keyboard

DIKEYBOARD_A…DIKEYBOARD_Z
: Letter keys A through Z on the main keyboard

DIKEYBOARD_ABNT_C1, DIKEYBOARD_ABNT_C2
: On numeric keypad of Brazilian keyboards

DIKEYBOARD_ADD
: PLUS SIGN (+) on the numeric keypad

DIKEYBOARD_APOSTROPHE
: APOSTROPHE (') key

DIKEYBOARD_APPS

DIKEYBOARD_AT
: On Japanese keyboard

DIKEYBOARD_AX
: On Japanese keyboard

DIKEYBOARD_BACK
BACKSPACE key

DIKEYBOARD_BACKSLASH
BACKSLASH (\) key

DIKEYBOARD_CALCULATOR
Dedicated key that launches a calculator

DIKEYBOARD_CAPITAL
CAPS LOCK key

DIKEYBOARD_COLON
On Japanese keyboard

DIKEYBOARD_COMMA
COMMA (,) key

DIKEYBOARD_CONVERT
On Japanese keyboard

DIKEYBOARD_DECIMAL
Decimal point (PERIOD) on the numeric keypad

DIKEYBOARD_DELETE
DELETE key on the main keyboard (not DEL on the numeric keypad)

DIKEYBOARD_DIVIDE
FORWARD SLASH (/) on the numeric keypad

DIKEYBOARD_DOWN, DIKEYBOARD_UP
DOWN ARROW, UP ARROW on the main keyboard (not the arrow keys on the numeric keypad)

DIKEYBOARD_END
END key on the main keyboard (not END on the numeric keypad)

DIKEYBOARD_EQUALS
EQUAL SIGN on the main keyboard

DIKEYBOARD_ESCAPE
ESC key

DIKEYBOARD_F(*n*)
Function keys F1 through F15

DIKEYBOARD_GRAVE
Grave accent (`)

DIKEYBOARD_HOME
HOME key on the main keyboard (not HOME on the numeric keypad)

DIKEYBOARD_INSERT
INSERT key on the main keyboard (not INS on the numeric keypad)

DIKEYBOARD_KANA
On Japanese keyboard

DIKEYBOARD_KANJI
On Japanese keyboard

DIKEYBOARD_LBRACKET, DIKEYBOARD_RBRACKET

Left ([ )and right ( ]) SQUARE BRACKET keys

DIKEYBOARD_LCONTROL, DIKEYBOARD_RCONTROL
Left and right CTRL keys

DIKEYBOARD_LEFT, DIKEYBOARD_RIGHT
LEFT ARROW, RIGHT ARROW on the main keyboard (not the arrow keys on the numeric keypad)

DIKEYBOARD_LMENU, DIKEYBOARD_RMENU
Left and right ALT keys

DIKEYBOARD_LSHIFT, DIKEYBOARD_RSHIFT
Left and right SHIFT keys

DIKEYBOARD_LWIN, DIKEYBOARD_RWIN
Left and right Microsoft® Windows® logo keys

DIKEYBOARD_MAIL
Dedicated key to launch an e-mail program

DIKEYBOARD_MEDIASELECT
Media Select key, which displays a selection of supported media players on the system

DIKEYBOARD_MEDIASTOP
Stops multimedia play

DIKEYBOARD_MINUS
(HYPHEN) (-) (MINUS SIGN) on the main keyboard

DIKEYBOARD_MULTIPLY
Asterisk (*) on the numeric keypad

DIKEYBOARD_MUTE
Mutes sound output

DIKEYBOARD_MYCOMPUTER
Launches My Computer on applicable Windows systems

DIKEYBOARD_NEXT
PAGE DOWN key on the main keyboard (not PGDN on the numeric keypad)

DIKEYBOARD_NEXTTRACK
Advances to the next CD/DVD track

DIKEYBOARD_NOCONVERT
On Japanese keyboard

DIKEYBOARD_NUMLOCK
NUM LOCK key

DIKEYBOARD_NUMPAD(*n*)
Number keys on the numeric keypad

DIKEYBOARD_NUMPADCOMMA
COMMA (,)on the NEC PC98 numeric keypad

DIKEYBOARD_NUMPADENTER
ENTER on the numeric keypad

DIKEYBOARD_NUMPADEQUALS

EQUAL SIGN on the numeric keypad of the NEC PC98 keyboard

DIKEYBOARD_OEM_102
On British and German keyboards

DIKEYBOARD_PAUSE
PAUSE key

DIKEYBOARD_PERIOD
PERIOD on the main keyboard

DIKEYBOARD_PLAYPAUSE
Pauses multimedia play

DIKEYBOARD_POWER
System power key

DIKEYBOARD_PREVTRACK
Play the previous CD/DVD track; CIRCUMFLEX on Japanese keyboard

DIKEYBOARD_PRIOR
PAGE UP key on the main keyboard (not PGUP on the numeric keypad)

DIKEYBOARD_RETURN
ENTER on the main keyboard

DIKEYBOARD_SCROLL
SCROLL LOCK key

DIKEYBOARD_SEMICOLON
SEMICOLON (;) key

DIKEYBOARD_SLASH
FORWARD SLASH (/) on the main keyboard (not FORWARD SLASH on the numeric keypad)

DIKEYBOARD_SLEEP
Sends the system into sleep mode

DIKEYBOARD_SPACE
The SPACEBAR

DIKEYBOARD_STOP
On Japanese keyboard

DIKEYBOARD_SUBTRACT
MINUS SIGN on the numeric keypad

DIKEYBOARD_SYSRQ
SYSRQ key

DIKEYBOARD_TAB
TAB key

DIKEYBOARD_UNDERLINE
On Japanese keyboard

DIKEYBOARD_UNLABELED
On Japanese keyboard

DIKEYBOARD_VOLUMEDOWN
Decreases sound output level

DIKEYBOARD_VOLUMEUP
Increases sound output level

DIKEYBOARD_WAKE
Wakes the system out of sleep mode

DIKEYBOARD_WEBBACK
Navigates a browser back one page in the history list

DIKEYBOARD_WEBFAVORITES
Displays the Microsoft Internet Explorer Favorites list, the Windows Favorites folder, or the Netscape Bookmarks list

DIKEYBOARD_WEBFORWARD
Instructs a browser to move to the next page in the history list

DIKEYBOARD_WEBHOME
Instructs a browser to load the user's home page

DIKEYBOARD_WEBREFRESH
Instructs a browser to reload the current page

DIKEYBOARD_WEBSEARCH
Instructs a browser to launch a search engine

DIKEYBOARD_WEBSTOP
Instructs a browser to stop loading the current page

DIKEYBOARD_YEN
On Japanese keyboard

The constants in this enumeration correspond to the members of **CONST_DIKEYFLAGS**, but their names begin with DIK rather than DIKEYBOARD and they have different values.

CONST_DIKEYFLAGS

[#]Used to identify keys in keyboard data.

```
Enum CONST_DIKEYFLAGS
    DIK_0          =  11
    DIK_1          =   2
    DIK_2          =   3
    DIK_3          =   4
    DIK_4          =   5
    DIK_5          =   6
    DIK_6          =   7
    DIK_7          =   8
    DIK_8          =   9
    DIK_9          =  10
    DIK_A          =  30 (&H1E)
    DIK_ABNT_C1    = 115 (&H73)
```

IDH_CONST_DIKEYFLAGS_dinput_vb

```
DIK_ABNT_C2      = 126 (&H7E)
DIK_ADD          =  78 (&H4E)
DIK_APOSTROPHE   =  40 (&H28)
DIK_APPS         = 221 (&HDD)
DIK_AT           = 145 (&H91)
DIK_AX           = 150 (&H96)
DIK_B            =  48 (&H30)
DIK_BACK         =  14
DIK_BACKSLASH    =  43 (&H2B)
DIK_BACKSPACE    =  14
DIK_C            =  46 (&H2E)
DIK_CALCULATOR   = 161 (&HA1)
DIK_CAPITAL      =  58 (&H3A)
DIK_CAPSLOCK     =  58 (&H3A)
DIK_CIRCUMFLEX   = 144 (&H90)
DIK_COLON        = 146 (&H92)
DIK_COMMA        =  51 (&H33)
DIK_CONVERT      = 121 (&H79)
DIK_D            =  32 (&H20)
DIK_DECIMAL      =  83 (&H53)
DIK_DELETE       = 211 (&HD3)
DIK_DIVIDE       = 181 (&HB5)
DIK_DOWN         = 208 (&HD0)
DIK_DOWNARROW    = 208 (&HD0)
DIK_E            =  18 (&H12)
DIK_END          = 207 (&HCF)
DIK_EQUALS       =  13
DIK_ESCAPE       =   1
DIK_F            =  33 (&H21)
DIK_F1           =  59 (&H3B)
DIK_F2           =  60 (&H3C)
DIK_F3           =  61 (&H3D)
DIK_F4           =  62 (&H3E)
DIK_F5           =  63 (&H3F)
DIK_F6           =  64 (&H40)
DIK_F7           =  65 (&H41)
DIK_F8           =  66 (&H42)
DIK_F9           =  67 (&H43)
DIK_F10          =  68 (&H44)
DIK_F11          =  87 (&H57)
DIK_F12          =  88 (&H58)
DIK_F13          = 100 (&H64)
DIK_F14          = 101 (&H65)
DIK_F15          = 102 (&H66)
DIK_G            =  34 (&H22)
DIK_GRAVE        =  41 (&H29)
```

```
DIK_H           =  35 (&H23)
DIK_HOME        = 199 (&HC7)
DIK_I           =  23 (&H17)
DIK_INSERT      = 210 (&HD2)
DIK_J           =  36 (&H24)
DIK_K           =  37 (&H25)
DIK_KANA        = 112 (&H70)
DIK_KANJI       = 148 (&H94)
DIK_L           =  38 (&H26)
DIK_LALT        =  56 (&H38)
DIK_LBRACKET    =  26 (&H1A)
DIK_LCONTROL    =  29 (&H1D)
DIK_LEFT        = 203 (&HCB)
DIK_LEFTARROW   = 203 (&HCB)
DIK_LMENU       =  56 (&H38)
DIK_LSHIFT      =  42 (&H2A)
DIK_LWIN        = 219 (&HDB)
DIK_M           =  50 (&H32)
DIK_MAIL        = 236 (&HEC)
DIK_MEDIASELECT = 237 (&HED)
DIK_MEDIASTOP   = 164 (&HA4)
DIK_MINUS       =  12
DIK_MULTIPLY    =  55 (&H37)
DIK_MUTE        = 160 (&HA0)
DIK_MYCOMPUTER  = 235 (&HEB)
DIK_N           =  49 (&H31)
DIK_NEXT        = 209 (&HD1)
DIK_NEXTTRACK   = 153 (&H99)
DIK_NOCONVERT   = 123 (&H7B)
DIK_NUMLOCK     =  69 (&H45)
DIK_NUMPAD0     =  82 (&H52)
DIK_NUMPAD1     =  79 (&H4F)
DIK_NUMPAD2     =  80 (&H50)
DIK_NUMPAD3     =  81 (&H51)
DIK_NUMPAD4     =  75 (&H4B)
DIK_NUMPAD5     =  76 (&H4C)
DIK_NUMPAD6     =  77 (&H4D)
DIK_NUMPAD7     =  71 (&H47)
DIK_NUMPAD8     =  72 (&H48)
DIK_NUMPAD9     =  73 (&H49)
DIK_NUMPADCOMMA = 179 (&HB3)
DIK_NUMPADENTER = 156 (&H9C)
DIK_NUMPADEQUALS = 141 (&H8D)
DIK_NUMPADMINUS =  74 (&H4A)
DIK_NUMPADPERIOD =  83 (&H53)
DIK_NUMPADPLUS  =  78 (&H4E)
```

```
DIK_NUMPADSLASH  = 181 (&HB5)
DIK_NUMPADSTAR   =  55 (&H37)
DIK_O            =  24 (&H18)
DIK_OEM_102      =  86 (&H56)
DIK_P            =  25 (&H19)
DIK_PAUSE        = 197 (&HC5)
DIK_PERIOD       =  52 (&H34)
DIK_PGDN         = 209 (&HD1)
DIK_PGUP         = 201 (&HC9)
DIK_PLAYPAUSE    = 162 (&HA2)
DIK_POWER        = 222 (&HDE)
DIK_PREVTRACK    = 144 (&H90)
DIK_PRIOR        = 201 (&HC9)
DIK_Q            =  16 (&H10)
DIK_R            =  19 (&H13)
DIK_RALT         = 184 (&HB8)
DIK_RBRACKET     =  27 (&H1B)
DIK_RCONTROL     = 157 (&H9D)
DIK_RETURN       =  28 (&H1C)
DIK_RIGHT        = 205 (&HCD)
DIK_RIGHTARROW   = 205 (&HCD)
DIK_RMENU        = 184 (&HB8)
DIK_RSHIFT       =  54 (&H36)
DIK_RWIN         = 220 (&HDC)
DIK_S            =  31 (&H1F)
DIK_SCROLL       =  70 (&H46)
DIK_SEMICOLON    =  39 (&H27)
DIK_SLASH        =  53 (&H35)
DIK_SLEEP        = 223 (&HDF)
DIK_SPACE        =  57 (&H39)
DIK_STOP         = 149 (&H95)
DIK_SUBTRACT     =  74 (&H4A)
DIK_SYSRQ        = 183 (&HB7)
DIK_T            =  20 (&H14)
DIK_TAB          =  15
DIK_U            =  22 (&H16)
DIK_UNDERLINE    = 147 (&H93)
DIK_UNLABELED    = 151 (&H97)
DIK_UP           = 200 (&HC8)
DIK_UPARROW      = 200 (&HC8)
DIK_V            =  47 (&H2F)
DIK_VOLUMEDOWN   = 174 (&HAE)
DIK_VOLUMEUP     = 176 (&HB0)
DIK_W            =  17 (&H11)
DIK_WAKE         = 227 (&HE3)
DIK_WEBBACK      = 234 (&HEA)
```

```
    DIK_WEBFAVORITES = 230 (&HE6)
    DIK_WEBFORWARD   = 233 (&HE9)
    DIK_WEBHOME      = 178 (&HB2)
    DIK_WEBREFRESH   = 231 (&HE7)
    DIK_WEBSEARCH    = 229 (&HE5)
    DIK_WEBSTOP      = 232 (&HE8)
    DIK_X            =  45 (&H2D)
    DIK_Y            =  21 (&H15)
    DIK_YEN          = 125 (&H7D)
    DIK_Z            =  44 (&H2C)
End Enum
```

Constants

DIK_(*n*)
: Number keys 0 through 9 on the main keyboard

DIK_A…DIK_Z
: Letter keys A through Z on the main keyboard

DIK_ABNT_C1, DIK_ABNT_C2
: On numeric keypad of Brazilian keyboards

DIK_ADD
: PLUS SIGN (+) on the numeric keypad

DIK_APOSTROPHE
: APOSTROPHE (') key

DIK_APPS

DIK_AT
: On Japanese keyboard

DIK_AX
: On Japanese keyboard

DIK_BACK
: BACKSPACE key

DIK_BACKSLASH
: BACKSLASH (\) key

DIK_BACKSPACE
: BACKSPACE key (alias of DIK_BACKSLASH)

DIK_CALCULATOR
: Dedicated key that launches a calculator

DIK_CAPITAL
: CAPS LOCK key

DIK_CAPSLOCK
: CAPS LOCK key (alias of DIK_CAPITAL)

DIK_CIRCUMFLEX

On Japanese keyboard

DIK_COLON
On Japanese keyboard

DIK_COMMA
COMMA (,) key

DIK_CONVERT
On Japanese keyboard

DIK_DECIMAL
Decimal point (PERIOD) on the numeric keypad

DIK_DELETE
DELETE key on the main keyboard (not DEL on the numeric keypad)

DIK_DIVIDE
FORWARDSLASH (/) on the numeric keypad

DIK_DOWN, DIK_UP
DOWN ARROW, UP ARROW on main keyboard (not the arrow keys on the numeric keypad)

DIK_DOWNARROW, UPARROW
DOWN ARROW, UP ARROW on main keyboard (not the arrow keys on the numeric keypad) (aliases of DIK_DOWN and DIK_UP)

DIK_END
END key on the main keyboard (not END on the numeric keypad)

DIK_EQUALS
EQUAL SIGN on the main keyboard

DIK_ESCAPE
ESC key

DIK_F(*n*)
Function keys F1 through F15

DIK_GRAVE
Grave accent (`)

DIK_HOME
HOME key on the main keyboard (not HOME on the numeric keypad)

DIK_INSERT
INSERT key on the main keyboard (not INS on the numeric keypad)

DIK_KANA
On Japanese keyboard

DIK_KANJI
On Japanese keyboard

DIK_LALT, DIK RALT
Left and right ALT keys (alias of DIK_LMENU and DIK_RMENU)

DIK_LBRACKET, DIK_RBRACKET
Left ([ ) and right ( ]) SQUAREBRACKET keys

DIK_LCONTROL, DIK_RCONTROL

Left and right CTRL keys

DIK_LEFT, DIK_RIGHT
LEFT ARROW, RIGHT ARROW on the main keyboard (not the arrow keys on the numeric keypad)

DIK LEFTARROW, DIK_RIGHTARROW
LEFT ARROW, RIGHT ARROW on the main keyboard (not the arrow keys on the numeric keypad) (aliases of DIK_LEFT and DIK_RIGHT)

DIK_LMENU, DIK_RMENU
Left and right ALT keys

DIK_LSHIFT, DIK_RSHIFT
Left and right SHIFT keys

DIK_LWIN, DIK_RWIN
Left and right Microsoft® Windows® logo keys

DIK_MAIL
Dedicated key to launch an e-mail program

DIK_MEDIASELECT
Media Select key, which displays a selection of supported media players on the system

DIK_MEDIASTOP
Stops multimedia play

DIK_MINUS
MINUS SIGN (hyphen) on the main keyboard

DIK_MULTIPLY
Asterisk (*) on the numeric keypad

DIK_MUTE
Mutes sound output

DIK_MYCOMPUTER
Launches **My Computer** on applicable Windows systems

DIK_NEXT
PAGE DOWN key on the main keyboard (not PGDN on the numeric keypad)

DIK_NEXTTRACK
Advances to the next CD/DVD track

DIK_NOCONVERT
On Japanese keyboard

DIK_NUMLOCK
NUM LOCK key

DIK_NUMPAD(*n*)
Number keys on the numeric keypad

DIK_NUMPADCOMMA
COMMA on the NEC PC98 numeric keypad

DIK_NUMPADENTER
ENTER on the numeric keypad

DIK_NUMPADEQUALS
EQUAL SIGN on the numeric keypad of the NEC PC98 keyboard

DIK_NUMPADMINUS
MINUS SIGN on the numeric keypad (alias of DIK_SUBTRACT)

DIK_NUMPADPERIOD
Decimal point (PERIOD) on the numeric keypad (alias of DIK_DECIMAL)

DIK_NUMPADPLUS
PLUS SIGN (+) on the numeric keypad (alias of DIK_ADD)

DIK_NUMPADSLASH
FORWARD SLASH (/) on the numeric keypad (alias of DIK_DIVIDE)

DIK_NUMPADSTAR
ASTERISK (*) on the numeric keypad (alias of DIK_MULTIPLY)

DIK_OEM_102
On British and German keyboards

DIK_PAUSE
PAUSE key

DIK_PERIOD
PERIOD on the main keyboard (not the decimal point on the numeric keypad)

DIK_PGDN, DIK_PGUP
PAGE DOWN, PAGE UP keys on the main keyboard (not PGDN and PGUP on the numeric keypad) (aliases of DIK_NEXT and DIK_PRIOR)

DIK_PLAYPAUSE
Pauses multimedia play

DIK_POWER
System power key

DIK_PREVTRACK
Play the previous CD/DVD track

DIK_PRIOR
PAGE UP key on the main keyboard (not PGUP on the numeric keypad)

DIK_RETURN
ENTER on the main keyboard

DIK_SCROLL
SCROLL LOCK key

DIK_SEMICOLON
SEMICOLON key

DIK_SLASH
FORWARD SLASH (/) on the main keyboard (not FORWARD SLASH on the numeric keypad)

DIK_SLEEP
Sends the system into sleep mode

DIK_SPACE
The SPACEBAR

DIK_STOP
On Japanese keyboard

DIK_SUBTRACT
MINUS SIGN on the numeric keypad

DIK_SYSRQ
SYSRQ key

DIK_TAB
TAB key

DIK_UNDERLINE
On Japanese keyboard

DIK_UNLABELED
On Japanese keyboard

DIK_VOLUMEDOWN
Decreases sound output level

DIK_VOLUMEUP
Increases sound output level

DIK_WAKE
Wakes the system out of sleep mode

DIK_WEBBACK
Navigates a browser back one page in the history list

DIK_WEBFAVORITES
Displays the Internet Explorer Favorites list, the Windows Favorites folder, or the Netscape Bookmarks list

DIK_WEBFORWARD
Instructs a browser to move to the next page in the history list

DIK_WEBHOME
Instructs a browser to load the user's home page

DIK_WEBREFRESH
Instructs a browser to reload the current page

DIK_WEBSEARCH
Instructs a browser to launch a search engine

DIK_WEBSTOP
Instructs a browser to stop loading the current page

DIK_YEN
On Japanese keyboard

Remarks

The constants in this enumeration that are not aliases correspond to the members of **CONST_DIKEYBOARDFLAGS**, begin with DIK rather than DIKEYBOARD, and have different values.

CONST_DIMAPFLAGS

#Used in the **lSemantic** member of the **DIACTION** type to map an action to any matching control on the device.

These constants can be used to map an application action to a virtual control that is not defined in a genre. Such actions are mapped after genre-specific actions. If the mapper has already mapped all matching controls to genre-specific actions, the action is left unmapped.

Actions mapped with these constants are treated with equal priority. If a device has one x-axis and the action array specifies DIAXIS_ANY_1 and DIAXIS_ANY_X_1, the action mapped to the x-axis is the one that appears first in the action array.

```
Enum CONST_DIMAPFLAGS
   DIAXIS_ANY_1   = -16760319 (&HFF004201)
   DIAXIS_ANY_2   = -16760318 (&HFF004202)
   DIAXIS_ANY_3   = -16760317 (&HFF004203)
   DIAXIS_ANY_4   = -16760316 (&HFF004204)
   DIAXIS_ANY_A_1 = -16530943 (&HFF03C201)
   DIAXIS_ANY_A_2 = -16530942 (&HFF03C202)
   DIAXIS_ANY_B_1 = -16498175 (&HFF044201)
   DIAXIS_ANY_B_2 = -16498174 (&HFF044202)
   DIAXIS_ANY_C_1 = -16465407 (&HFF04C201)
   DIAXIS_ANY_C_2 = -16465406 (&HFF04C202)
   DIAXIS_ANY_R_1 = -16629247 (&HFF024201)
   DIAXIS_ANY_R_2 = -16629246 (&HFF024202)
   DIAXIS_ANY_S_1 = -16432639 (&HFF054201)
   DIAXIS_ANY_S_2 = -16432638 (&HFF054202)
   DIAXIS_ANY_U_1 = -16596479 (&HFF02C201)
   DIAXIS_ANY_U_2 = -16596478 (&HFF02C202)
   DIAXIS_ANY_V_1 = -16563711 (&HFF034201)
   DIAXIS_ANY_V_2 = -16563710 (&HFF034202)
   DIAXIS_ANY_X_1 = -16727551 (&HFF00C201)
   DIAXIS_ANY_X_2 = -16727550 (&HFF00C202)
   DIAXIS_ANY_Y_1 = -16694783 (&HFF014201)
   DIAXIS_ANY_Y_2 = -16694782 (&HFF014202)
   DIAXIS_ANY_Z_1 = -16662015 (&HFF01C201)
   DIAXIS_ANY_Z_2 = -16662014 (&HFF01C202)
   DIBUTTON_ANY   = -16759808 (&HFF004400)
   DIPOV_ANY_1    = -16759295 (&HFF004601)
   DIPOV_ANY_2    = -16759294 (&HFF004602)
   DIPOV_ANY_3    = -16759293 (&HFF004603)
   DIPOV_ANY_4    = -16759292 (&HFF004604)
End Enum
```

IDH_CONST_DIMAPFLAGS_dinput_vb

Constants

DIAXIS_ANY_1
Any axis
DIAXIS_ANY_2
Any axis
DIAXIS_ANY_3
Any axis
DIAXIS_ANY_4
Any axis
DIAXIS_ANY_A_1
Any accelerator
DIAXIS_ANY_A_2
Any accelerator
DIAXIS_ANY_B_1
Any brake
DIAXIS_ANY_B_2
Any brake
DIAXIS_ANY_C_1
Any clutch
DIAXIS_ANY_C_2
Any clutch
DIAXIS_ANY_R_1
Any r-axis
DIAXIS_ANY_R_2
Any r-axis
DIAXIS_ANY_S_1
Any s-axis
DIAXIS_ANY_S_2
Any s-axis
DIAXIS_ANY_U_1
Any u-axis
DIAXIS_ANY_U_2
Any u-axis
DIAXIS_ANY_V_1
Any v-axis
DIAXIS_ANY_V_2
Any v-axis
DIAXIS_ANY_X_1
Any x-axis
DIAXIS_ANY_X_1

Any x-axis

DIAXIS_ANY_Y_1
Any y-axis

DIAXIS_ANY_Y_1
Any y-axis

DIAXIS_ANY_Z_1
Any z-axis

DIAXIS_ANY_Z_1
Any z-axis

DIBUTTON_ANY
Any button

DIPOV_ANY_1
Any point-of-view controller

DIPOV_ANY_2
Any point-of-view controller

DIPOV_ANY_3
Any point-of-view controller

DIPOV_ANY_4
Any point-of-view controller

CONST_DIMOUSEFLAGS

#Used to specify mouse buttons and axes in the **lSemantic** member of the **DIACTION** type.

```
Enum CONST_DIMOUSEFLAGS
   DIMOUSE_BUTTON0 = -2113928180 (&H8200040C)
   DIMOUSE_BUTTON1 = -2113928179 (&H8200040D)
   DIMOUSE_BUTTON2 = -2113928178 (&H8200040E)
   DIMOUSE_BUTTON3 = -2113928177 (&H8200040F)
   DIMOUSE_BUTTON4 = -2113928176 (&H82000410)
   DIMOUSE_BUTTON5 = -2113928175 (&H82000411)
   DIMOUSE_BUTTON6 = -2113928174 (&H82000412
   DIMOUSE_BUTTON7 = -2113928173 (&H82000413)
   DIMOUSE_WHEEL   = -2113928440 (&H82000308)
   DIMOUSE_XAXIS   = -2113928448 (&H82000300)
   DIMOUSE_XAXISAB = -2113928704 (&H82000200)
   DIMOUSE_YAXIS   = -2113928444 (&H82000304)
   DIMOUSE_YAXISAB = -2113928700 (&H82000204)
End Enum
```

IDH_CONST_DIMOUSEFLAGS_dinput_vb

Constants

DIMOUSE_BUTTON(*n*)
Buttons 0 through 7.

DIMOUSE_WHEEL, DIMOUSE_XAXIS, DIMOUSE_YAXIS
Axes that return relative data.

DIMOUSE_XAXISAB, DIMOUSE_YAXISAB
Axes that return absolute data.

CONST_DIMOUSEOFS

#Used to identify buttons and axes in mouse data. The values of the constants are the offset of the data within the data format for immediate data. The same identifiers are used in buffered data packets.

```
Enum CONST_DIMOUSEOFS
   DIMOFS_BUTTON0 = 12
   DIMOFS_BUTTON1 = 13
   DIMOFS_BUTTON2 = 14
   DIMOFS_BUTTON3 = 15
   DIMOFS_BUTTON4 = 16 (&H10)
   DIMOFS_BUTTON5 = 17 (&H11)
   DIMOFS_BUTTON6 = 18 (&H12)
   DIMOFS_BUTTON7 = 19 (&H13)
   DIMOFS_X      = 0
   DIMOFS_Y      = 4
   DIMOFS_Z      = 8
End Enum
```

Constants

DIMOFS_BUTTON(*n*)
Offset of the data for buttons 0 through 7.

DIMOFS_X, DIMOFS_Y, DIMOFS_Z
Offset of the data for the axis.

CONST_DINPUT

#The **CONST_DINPUT** enumeration contains various constants that are used throughout Microsoft® DirectInput®.

```
Enum CONST_DINPUT
   DIEB_NOTRIGGER            =        -1 (&HFFFFFFFF)
   DIPROPAUTOCENTER_OFF      =         0
```

IDH_CONST_DIMOUSEOFS_dinput_vb
IDH_CONST_DINPUT_dinput_vb

```
    DIPROPAUTOCENTER_ON        =         1
    DIPROPAXISMODE_ABS         =         0
    DIPROPAXISMODE_REL         =         1
    DIPROPCALIBRATIONMODE_COOKED =         0
    DIPROPCALIBRATIONMODE_RAW    =         1
    DIPROPRANGE_NOMAX          =  2147483647 (&H7FFFFFFF)
    DIPROPRANGE_NOMIN          = -2147483648 (&H80000000)
End Enum
```

Constants

DIEB_NOTRIGGER
: Used in the **lTriggerButton** member of the **DIEFFECT** type to specify that the effect has no trigger button.

DIPROPAUTOCENTER_OFF
: The device should not automatically center when the user releases it. See **DirectInputDevice8.SetProperty**.

DIPROPAUTOCENTER_ON
: The device should automatically center when the user releases it. See **DirectInputDevice8.SetProperty**.

DIPROPAXISMODE_ABS
: Used in **DirectInputDevice8.GetProperty** and **DirectInputDevice8.SetProperty** to represent absolute axis mode.

DIPROPAXISMODE_REL
: Used in **DirectInputDevice8.GetProperty** and **DirectInputDevice8.SetProperty** to represent relative axis mode.

DIPROPCALIBRATIONMODE_COOKED
: Used in **DirectInputDevice8.SetProperty** to set the DIPROP_CALIBRATIONMODE property.

DIPROPCALIBRATIONMODE_RAW
: Used in **DirectInputDevice8.SetProperty** to set the DIPROP_CALIBRATIONMODE property.

DIPROPRANGE_NOMAX
: Returned from **DirectInputDevice8.GetProperty** if the axis has no upper limit on its range.

DIPROPRANGE_NOMIN
: Returned from **DirectInputDevice8.GetProperty** if the axis has no lower limit on its range.

CONST_DINPUTERR

#The **CONST_DINPUTERR** enumeration contains the error codes for Microsoft® DirectInput®. All the error codes and definitions can be found in the Error Codes topic.

IDH_CONST_DINPUTERR_dinput_vb

```
Enum CONST_DINPUTERR
   DI_OK                    =          0
   DIERR_ACQUIRED           = -2147024726 (&H800700AA)
   DIERR_ALREADYINITIALIZED     = -2147023649 (&H800704DF)
   DIERR_BADDRIVERVER           = -2147024777 (&H80070077)
   DIERR_BETADIRECTINPUTVERSION  = -2147023743 (&H80070481)
   DIERR_DEVICEFULL             = -2147220991 (&H80040201)
   DIERR_DEVICENOTREG           = -2147221164 (&H80040154)
   DIERR_EFFECTPLAYING          = -2147220984 (&H80040208)
   DIERR_GENERIC              = -2147467259 (&H80004005)
   DIERR_HANDLEEXISTS           = -2147024891 (&H80070005)
   DIERR_HASEFFECTS             = -2147220988 (&H80040204)
   DIERR_INCOMPLETEEFFECT       = -2147220986 (&H80040206)
   DIERR_INPUTLOST              = -2147024866 (&H8007001E)
   DIERR_INVALIDHANDLE          = -2147024890 (&H80070006)
   DIERR_INVALIDPARAM           =          5
   DIERR_MOREDATA               = -2147220990 (&H80040202)
   DIERR_NOAGGREGATION          = -2147467262 (&H80004002)
   DIERR_NOINTERFACE            =        430 (&H1AE)
   DIERR_NOTACQUIRED            = -2147024884 (&H8007000C)
   DIERR_NOTBUFFERED            = -2147220985 (&H80040207)
   DIERR_NOTDOWNLOADED          = -2147220989 (&H80040203)
   DIERR_NOTEXCLUSIVEACQUIRED    = -2147220987 (&H80040205)
   DIERR_NOTFOUND               = -2147024894 (&H80070002)
   DIERR_NOTINITIALIZED         = -2147024875 (&H80070015)
   DIERR_OBJECTNOTFOUND         = -2147024894 (&H80070002)
   DIERR_OLDDIRECTINPUTVERSION   = -2147023746 (&H8007047E)
   DIERR_OTHERAPPHASPRIO        = -2147024891 (&H80070005)
   DIERR_OUTOFMEMORY            =          7
   DIERR_READONLY               = -2147024891 (&H80070005)
   DIERR_REPORTFULL             = -2147220982 (&H8004020A)
   DIERR_UNPLUGGED              = -2147220983 (&H80040209)
   DIERR_UNSUPPORTED            =        445 (&H1BD)
   E_PENDING                  = -2147024889 (&H80070007)
End Enum
```

Constants

DI_OK
: Success.

DIERR_ACQUIRED
: The operation cannot be performed while the device is acquired.

DIERR_ALREADYINITIALIZED
: This object is already initialized.

DIERR_BADDRIVERVER

The object could not be created due to an incompatible driver version or mismatched or incomplete driver components.

DIERR_BETADIRECTINPUTVERSION
The application was written for an unsupported prerelease version of DirectInput.

DIERR_DEVICEFULL
The device is full.

DIERR_DEVICENOTREG
The device or device instance is not registered with DirectInput. This value is equal to the REGDB_E_CLASSNOTREG standard COM return value.

DIERR_EFFECTPLAYING
The parameters were updated in memory but were not downloaded to the device because the device does not support updating an effect while it is still playing.

DIERR_GENERIC
An undetermined error occurred inside the DirectInput subsystem. This value is equal to the E_FAIL standard COM return value.

DIERR_HANDLEEXISTS
The device already has an event notification associated with it. This value is equal to the E_ACCESSDENIED standard COM return value.

DIERR_HASEFFECTS
The device cannot be reinitialized because effects are attached to it.

DIERR_INCOMPLETEEFFECT
The effect could not be downloaded because essential information is missing. For example, no axes have been associated with the effect, or no type-specific information has been supplied.

DIERR_INPUTLOST
Access to the input device has been lost. It must be reacquired.

DIERR_INVALIDHANDLE
An invalid window handle was passed to the method.

DIERR_INVALIDPARAM
An invalid parameter was passed to the returning function, or the object was not in a state that permitted the function to be called. This value is equal to the E_INVALIDARG standard COM return value.

DIERR_MOREDATA
Not all the requested information fitted into the buffer.

DIERR_NOAGGREGATION
This object does not support aggregation.

DIERR_NOINTERFACE
The specified interface is not supported by the object. This value is equal to the E_NOINTERFACE standard COM return value.

DIERR_NOTACQUIRED
The operation cannot be performed unless the device is acquired.

DIERR_NOTBUFFERED

The device is not buffered. Set the DIPROP_BUFFERSIZE property to enable buffering.

DIERR_NOTDOWNLOADED
The effect is not downloaded.

DIERR_NOTEXCLUSIVEACQUIRED
The operation cannot be performed unless the device is acquired in DISCL_EXCLUSIVE mode.

DIERR_NOTFOUND
The requested object does not exist.

DIERR_NOTINITIALIZED
The object has not been initialized.

DIERR_OBJECTNOTFOUND
The requested object does not exist.

DIERR_OLDDIRECTINPUTVERSION
The application requires a newer version of DirectInput.

DIERR_OTHERAPPHASPRIO
Another application has a higher priority level, preventing this call from succeeding. This value is equal to the E_ACCESSDENIED standard COM return value. This error can be returned when an application has only foreground access to a device but is attempting to acquire the device while in the background.

DIERR_OUTOFMEMORY
The DirectInput subsystem couldn't allocate sufficient memory to complete the call. This value is equal to the E_OUTOFMEMORY standard COM return value.

DIERR_READONLY
The specified property cannot be changed. This value is equal to the E_ACCESSDENIED standard COM return value.

DIERR_REPORTFULL
More information was requested to be sent than can be sent to the device.

DIERR_UNPLUGGED
The operation could not be completed because the device is not plugged in.

DIERR_UNSUPPORTED
The function called is not supported at this time. This value is equal to the E_NOTIMPL standard COM return value.

E_PENDING
Data is not yet available.

CONST_DIPHFLAGS

#Members of the **CONST_DIPHFLAGS** enumeration are used to specify how a device object is identified. They are used in the **DirectInputDevice8.GetObjectInfo** method as well as in the **DIPROPLONG** and **DIPROPRANGE** types.

Enum CONST_DIPHFLAGS

IDH_CONST_DIPHFLAGS_dinput_vb

```
    DIPH_BYID     = 2
    DIPH_BYOFFSET = 1
    DIPH_BYUSAGE  = 3
    DIPH_DEVICE   = 0
End Enum
```

Constants

DIPH_BYID
: The device object is identified by the instance identifier obtained from the return value of the **DirectInputDeviceObjectInstance.GetType** method.

DIPH_BYOFFSET
: The device object is identified by the offset into the current data format of the object whose information is being accessed.

DIPH_BYUSAGE
: The device object is identified by the HID usage page and usage values in packed form.

DIPH_DEVICE
: The property applies to the entire device, not to a particular object.

CONST_DISCLFLAGS

#The **CONST_DISCLFLAGS** enumeration is used in the *flags* parameter of the **DirectInputDevice8.SetCooperativeLevel** method to determine how the device interacts with other instances of the device and the rest of the system.

```
Enum CONST_DISCLFLAGS
    DISCL_BACKGROUND   =  8
    DISCL_EXCLUSIVE    =  1
    DISCL_FOREGROUND   =  4
    DISCL_NONEXCLUSIVE =  2
    DISCL_NOWINKEY     = 16 (&H10)
End Enum
```

Constants

DISCL_BACKGROUND
: The application requires background access. If background access is granted, the device may be acquired at any time, even when the associated window is not the active window.

DISCL_EXCLUSIVE
: The application requires exclusive access. If exclusive access is granted, no other instance of the device may obtain exclusive access to the device while it is acquired. Note, however, non-exclusive access to the device is always permitted, even if another application has obtained exclusive access.

IDH_CONST_DISCLFLAGS_dinput_vb

If an application acquires the mouse or keyboard device in exclusive mode, the user will not be able to use the window menu or move and resize the window.

DISCL_FOREGROUND
The application requires foreground access. If foreground access is granted, the device is automatically unacquired when the associated window moves to the background.

DISCL_NONEXCLUSIVE
The application requires non-exclusive access. Access to the device will not interfere with other applications that are accessing the same device.

DISCL_NOWINKEY
The Microsoft® Windows® logo key is disabled so that users cannot inadvertently break out of the application. This value can be combined with DISCL_NONEXCLUSIVE. In exclusive mode, the Windows logo key is always disabled. Note, however, that DISCL_NOWINKEY has no effect when the default action mapping UI is displayed, and the Windows logo key operates normally as long as that UI is present.

Applications must specify either DISCL_FOREGROUND or DISCL_BACKGROUND; it is an error to specify both or neither. Similarly, applications must specify either DISCL_EXCLUSIVE or DISCL_NONEXCLUSIVE.

CONST_DISDDFLAGS

#Members of the **CONST_DISDDFLAGS** enumeration are passed to the **DirectInputDevice8.SendDeviceData** method.

```
Enum CONST_DISDDFLAGS
   DISDD_CONTINUE = 1
   DISDD_DEFAULT  = 0
End Enum
```

Constants

DISDD_CONTINUE
Data will overlaid on existing data. For more information, see the Remarks section for **DirectInputDevice8.SendDeviceData** method.

DISDD_DEFAULT
Data will not be overlaid on existing data.

IDH_CONST_DISDDFLAGS_dinput_vb

CONST_DISFFCFLAGS

#Members of the **CONST_DISFFCFLAGS** enumeration are used by the **DirectInputDevice8.SendForceFeedbackCommand** method to specify the command.

```
Enum CONST_DISFFCFLAGS
   DISFFC_CONTINUE      = 8
   DISFFC_PAUSE         = 4
   DISFFC_RESET         = 1
   DISFFC_SETACTUATORSOFF = 32 (&H20)
   DISFFC_SETACTUATORSON  = 16 (&H10)
   DISFFC_STOPALL       = 2
End Enum
```

Constants

DISFFC_CONTINUE
: Paused playback of all active effects is to be continued. It is an error to send this command when the device is not in a paused state.

DISFFC_PAUSE
: Playback of all active effects is to be paused. This command also stops the clock on effects, so that they continue playing to their full duration when restarted.

  While the device is paused, new effects may not be started and existing ones may not be modified. Doing so may result in the subsequent DISFFC_CONTINUE command failing to perform properly.

  To abandon a pause and stop all effects, use the DISFFC_STOPALL or DISFCC_RESET commands.

DISFFC_RESET
: The device's force-feedback system is to be put in its startup state. All effects are removed from the device, are no longer valid, and must be recreated if they are to be used again. The device's actuators are disabled.

DISFFC_SETACTUATORSOFF
: The device's force-feedback actuators are to be disabled. While the actuators are off, effects continue to play but are ignored by the device. Using the analogy of a sound playback device, they are muted rather than paused.

DISFFC_SETACTUATORSON
: The device's force-feedback actuators are to be enabled.

DISFFC_STOPALL
: Playback of any active effects is to be stopped. All active effects will be reset, but are still being maintained by the device and are still valid. If the device is in a paused state, that state is lost.

  This command is equivalent to calling the **DirectInputEffect.Stop** method for each effect playing.

IDH_CONST_DISFFCFLAGS_dinput_vb

CONST_DIVOICEFLAGS

#Used to specify voice device objects in the **lSemantic** member of the **DIACTION** type.

```
Enum CONST_DIVOICEFLAGS
   DIVOICE_ALL         = -2097150966 (&H8300040A)
   DIVOICE_CHANNEL1    = -2097150975 (&H83000401)
   DIVOICE_CHANNEL2    = -2097150974 (&H83000402)
   DIVOICE_CHANNEL3    = -2097150973 (&H83000403)
   DIVOICE_CHANNEL4    = -2097150972 (&H83000404)
   DIVOICE_CHANNEL5    = -2097150971 (&H83000405)
   DIVOICE_CHANNEL6    = -2097150970 (&H83000406)
   DIVOICE_CHANNEL7    = -2097150969 (&H83000407)
   DIVOICE_CHANNEL8    = -2097150968 (&H83000408)
   DIVOICE_PLAYBACKMUTE = -2097150964 (&H8300040C)
   DIVOICE_RECORDMUTE   = -2097150965 (&H8300040B)
   DIVOICE_TEAM        = -2097150967 (&H83000409)
   DIVOICE_TRANSMIT    = -2097150963 (&H8300040D)
   DIVOICE_VOICECOMMAND = -2097150960 (&H83000410)
End Enum
```

DIVOICE_ALL
: Broadcast voice input on all channels

DIVOICE_CHANNEL(*n*)
: Individual broadcast channels 1 through 8

DIVOICE_PLAYBACKMUTE
: Mute voice broadcast

DIVOICE_RECORDMUTE
: Mute recording

DIVOICE_TEAM
: Broadcast voice only to team members

DIVOICE_TRANSMIT
: Send broadcast

DIVOICE_VOICECOMMAND
: Accept voice input as voice command rather than broadcast data

Keyboard Keys

This section contains information on the following topics:

IDH_CONST_DIVOICEFLAGS_dinput_vb

- Keyboard Device Constants
- DirectInput and Japanese Keyboards

Keyboard Device Constants

#Members of the **CONST_DIKEYFLAGS** enumeration represent offsets within a keyboard device's data packet, a 256-byte array. The data at a given offset is associated with a keyboard key.

The standard keyboard device constants are listed in the following table in ascending order of data offset.

The **CONST_DIKEYBOARDFLAGS** enumeration, used in action mapping, also contains constants for each of the keys in the table. The names of these constants begin with DIKEYBOARD rather than DIK.

Constant	Note
DIK_ESCAPE	
DIK_1	On main keyboard
DIK_2	On main keyboard
DIK_3	On main keyboard
DIK_4	On main keyboard
DIK_5	On main keyboard
DIK_6	On main keyboard
DIK_7	On main keyboard
DIK_8	On main keyboard
DIK_9	On main keyboard
DIK_0	On main keyboard
DIK_MINUS	On main keyboard
DIK_EQUALS	On main keyboard
DIK_BACK	BACKSPACE
DIK_TAB	
DIK_Q	
DIK_W	
DIK_E	
DIK_R	
DIK_T	
DIK_Y	
DIK_U	

IDH_Keyboard_Device_Constants_dinput_vb

DIK_I	
DIK_O	
DIK_P	
DIK_LBRACKET	Left square bracket.
DIK_RBRACKET	Right square bracket
DIK_RETURN	ENTER on main keyboard
DIK_LCONTROL	Left CTRL
DIK_A	
DIK_S	
DIK_D	
DIK_F	
DIK_G	
DIK_H	
DIK_J	
DIK_K	
DIK_L	
DIK_SEMICOLON	
DIK_APOSTROPHE	
DIK_GRAVE	Grave accent (`)
DIK_LSHIFT	Left SHIFT
DIK_BACKSLASH	
DIK_Z	
DIK_X	
DIK_C	
DIK_V	
DIK_B	
DIK_N	
DIK_M	
DIK_COMMA	
DIK_PERIOD	On main keyboard
DIK_SLASH	Forward slash on main keyboard
DIK_RSHIFT	Right SHIFT key
DIK_MULTIPLY	Asterisk on numeric keypad
DIK_LMENU	Left ALT
DIK_SPACE	SPACEBAR
DIK_CAPITAL	CAPS LOCK
DIK_F1	
DIK_F2	

DIK_F3	
DIK_F4	
DIK_F5	
DIK_F6	
DIK_F7	
DIK_F8	
DIK_F9	
DIK_F10	
DIK_NUMLOCK	
DIK_SCROLL	SCROLL LOCK
DIK_NUMPAD7	
DIK_NUMPAD8	
DIK_NUMPAD9	
DIK_SUBTRACT	MINUS SIGN on numeric keypad
DIK_NUMPAD4	
DIK_NUMPAD5	
DIK_NUMPAD6	
DIK_ADD	PLUS SIGN on numeric keypad
DIK_NUMPAD1	
DIK_NUMPAD2	
DIK_NUMPAD3	
DIK_NUMPAD0	
DIK_DECIMAL	PERIOD (decimal point) on numeric keypad
DIK_OEM_102	On British and German keyboards.
DIK_F11	
DIK_F12	
DIK_F13	
DIK_F14	
DIK_F15	
DIK_KANA	On Japanese keyboard
DIK_ABNT_C1	On numeric pad of Brazilian keyboards
DIK_CONVERT	On Japanese keyboard
DIK_NOCONVERT	On Japanese keyboard
DIK_YEN	On Japanese keyboard
DIK_ABNT_C2	On numeric pad of Brazilian keyboards
DIK_NUMPADEQUALS	On numeric keypad (NEC PC98)
DIK_PREVTRACK	Previous track; circumflex on Japanese keyboard
DIK_AT	On Japanese keyboard

DIK_COLON	On Japanese keyboard
DIK_UNDERLINE	On Japanese keyboard
DIK_KANJI	On Japanese keyboard
DIK_STOP	On Japanese keyboard
DIK_AX	On Japanese keyboard
DIK_UNLABELED	On Japanese keyboard
DIK_NEXTTRACK	Next track
DIK_NUMPADENTER	
DIK_RCONTROL	Right CTRL
DIK_MUTE	
DIK_CALCULATOR	
DIK_PLAYPAUSE	
DIK_MEDIASTOP	
DIK_VOLUMEDOWN	
DIK_VOLUMEUP	
DIK_WEBHOME	
DIK_NUMPADCOMMA	COMMA on NEC PC98 numeric keypad
DIK_DIVIDE	Forward slash on numeric keypad
DIK_SYSRQ	
DIK_RMENU	Right ALT
DIK_PAUSE	
DIK_HOME	
DIK_UP	UP ARROW
DIK_PRIOR	PAGE UP
DIK_LEFT	LEFT ARROW
DIK_RIGHT	RIGHT ARROW
DIK_END	
DIK_DOWN	DOWN ARROW
DIK_NEXT	PAGE DOWN
DIK_INSERT	
DIK_DELETE	
DIK_LWIN	Left Microsoft® Windows® logo key
DIK_RWIN	Right Windows logo key
DIK_APPS	
DIK_POWER	
DIK_SLEEP	
DIK_WAKE	
DIK_WEBSEARCH	

DIK_WEBFAVORITES
DIK_WEBREFRESH
DIK_WEBSTOP
DIK_WEBFORWARD
DIK_WEBBACK
DIK_MYCOMPUTER
DIK_MAIL
DIK_MEDIASELECT

For information on Japanese keyboards, see DirectInput and Japanese Keyboards.

DirectInput and Japanese Keyboards

#The following chart lists keys that are available on Japanese keyboards but not on standard U.S. keyboards, as well as keys that are available on U.S. keyboards but not on the various Japanese keyboards.

On Microsoft® Windows® 2000 and with some keyboards (such as the NEC PC-98) using other operating systems, the DIK_CAPSLOCK, DIK_KANJI, and DIK_KANA keys are toggle buttons and not push buttons. These generate a Down event when first pressed, then generate an Up event when pressed a second time.

Japanese Keyboard	Additional Keys	Missing Keys
DOS/V 106, NEC PC-98 106	DIK_AT, DIK_CIRCUMFLEX, DIK_COLON, DIK_CONVERT, DIK_KANA, DIK_KANJI, DIK_NOCONVERT, DIK_YEN	DIK_APOSTROPHE, DIK_EQUALS, DIK_GRAVE
NEC PC-98	DIK_AT, DIK_CIRCUMFLEX, DIK_COLON, DIK_F13, DIK_F14, DIK_F15, DIK_KANA, DIK_KANJI, DIK_NOCONVERT, DIK_NUMPADCOMMA, DIK_NUMPADEQUALS, DIK_STOP,	DIK_APOSTROPHE, DIK_BACKSLASH, DIK_EQUALS, DIK_GRAVE, DIK_NUMLOCK, DIK_NUMPADENTER, DIK_RCONTROL, DIK_RMENU, DIK_RSHIFT,

IDH_DirectInput_and_Japanese_Keyboards_dinput_vb

	DIK_UNDERLINE, DIK_YEN	DIK_SCROLL
AX	DIK_AX, DIK_CONVERT, DIK_KANJI, DIK_NOCONVERT, DIK_YEN	DIK_RCONTROL, DIK_RMENU
J-3100	DIK_KANA, DIK_KANJI, DIK_NOLABEL, DIK_YEN	DIK_RCONTROL, DIK_RMENU

Action Mapping Constants

This section is a reference to the constants defined in the CONST_DIGENRE enumeration for application genres and controls mapped to actions within each genre.

The control constants, such as DIAXIS_FIGHTINGH_MOVE, are used in the **ActionName** member of the **DIACTION** structure to associate the control with an action in the application.

The constants for the genres, such as DIVIRTUAL_FIGHTING_HAND2HAND, are used in the **lGenre** member of the **DIACTIONFORMAT** structure.

Action mapping constants are provided for application genres in the following categories.

- Action Genres
- Arcade Genres
- CAD Genres
- Control Genres
- Driving Genres
- Flight Genres
- Sports Genres
- Strategy Genres

The **ActionName** member of the **DIACTION** structure member can map the action to a particular device object rather than a virtual control. Device object constants are provided in the following categories.

- Keyboard Mapping Constants
- Mouse Mapping Constants
- DirectPlay Voice Mapping Constants
- Any-Control Constants

Action Genres

The following genres contain controls for fighting and action games.

- Hand-to-Hand
- Shooting
- Third-Person Action

Hand-to-Hand

The DIVIRTUAL_FIGHTING_HAND2HAND genre contains controls for a first-person fighting game without guns.

Priority 1 Controls

DIAXIS_FIGHTINGH_LATERAL
DIAXIS_FIGHTINGH_MOVE
DIBUTTON_FIGHTINGH_BLOCK
DIBUTTON_FIGHTINGH_CROUCH
DIBUTTON_FIGHTINGH_JUMP
DIBUTTON_FIGHTINGH_KICK
DIBUTTON_FIGHTINGH_MENU
DIBUTTON_FIGHTINGH_PUNCH
DIBUTTON_FIGHTINGH_SPECIAL1
DIBUTTON_FIGHTINGH_SPECIAL2

Priority 2 Controls

DIAXIS_FIGHTINGH_ROTATE
DIBUTTON_FIGHTINGH_BACKWARD_LINK
DIBUTTON_FIGHTINGH_DEVICE
DIBUTTON_FIGHTINGH_DISPLAY
DIBUTTON_FIGHTINGH_DODGE
DIBUTTON_FIGHTINGH_FORWARD_LINK
DIBUTTON_FIGHTINGH_LEFT_LINK
DIBUTTON_FIGHTINGH_PAUSE
DIBUTTON_FIGHTINGH_RIGHT_LINK
DIBUTTON_FIGHTINGH_SELECT
DIHATSWITCH_FIGHTINGH_SLIDE

Shooting

The DIVIRTUAL_FIGHTING_FPS genre contains controls for a first-person fighting game with guns.

Priority 1 Controls

DIAXIS_FPS_LOOKUPDOWN
DIAXIS_FPS_MOVE
DIAXIS_FPS_ROTATE
DIBUTTON_FPS_APPLY
DIBUTTON_FPS_CROUCH
DIBUTTON_FPS_FIRE
DIBUTTON_FPS_JUMP
DIBUTTON_FPS_MENU
DIBUTTON_FPS_SELECT
DIBUTTON_FPS_STRAFE
DIBUTTON_FPS_WEAPONS

Priority 2 Controls

DIAXIS_FPS_SIDESTEP
DIBUTTON_FPS_BACKWARD_LINK
DIBUTTON_FPS_DEVICE
DIBUTTON_FPS_DISPLAY
DIBUTTON_FPS_DODGE
DIBUTTON_FPS_FIRESECONDARY
DIBUTTON_FPS_FORWARD_LINK
DIBUTTON_FPS_GLANCE_DOWN_LINK
DIBUTTON_FPS_GLANCE_UP_LINK
DIBUTTON_FPS_GLANCEL
DIBUTTON_FPS_GLANCER
DIBUTTON_FPS_PAUSE
DIBUTTON_FPS_ROTATE_LEFT_LINK
DIBUTTON_FPS_ROTATE_RIGHT_LINK
DIHATSWITCH_FPS_GLANCE

Third-Person Action

The DIVIRTUAL_FIGHTING_THIRDPERSON genre contains controls for a third-person action game.

Priority 1 Controls

DIAXIS_TPS_MOVE
DIAXIS_TPS_TURN
DIBUTTON_TPS_ACTION
DIBUTTON_TPS_JUMP
DIBUTTON_TPS_MENU
DIBUTTON_TPS_RUN
DIBUTTON_TPS_SELECT
DIBUTTON_TPS_USE

Priority 2 Controls
DIAXIS_TPS_STEP
DIBUTTON_TPS_BACKWARD_LINK
DIBUTTON_TPS_DEVICE
DIBUTTON_TPS_DODGE
DIBUTTON_TPS_FORWARD_LINK
DIBUTTON_TPS_GLANCE_DOWN_LINK
DIBUTTON_TPS_GLANCE_LEFT_LINK
DIBUTTON_TPS_GLANCE_RIGHT_LINK
DIBUTTON_TPS_GLANCE_UP_LINK
DIBUTTON_TPS_INVENTORY
DIBUTTON_TPS_PAUSE
DIBUTTON_TPS_STEPLEFT
DIBUTTON_TPS_STEPRIGHT
DIBUTTON_TPS_TURN_LEFT_LINK
DIBUTTON_TPS_TURN_RIGHT_LINK
DIBUTTON_TPS_VIEW
DIHATSWITCH_TPS_GLANCE

Arcade Genres

The following genres contain controls for arcade-type games in which the object is to move a figure through a dangerous environment.

- Platform
- Side-to-Side

Platform

The DIVIRTUAL_ARCADE_PLATFORM genre contains controls for an arcade-style game.

Priority 1 Controls

DIAXIS_ARCADEP_LATERAL
DIAXIS_ARCADEP_MOVE
DIBUTTON_ARCADEP_CROUCH
DIBUTTON_ARCADEP_FIRE
DIBUTTON_ARCADEP_JUMP
DIBUTTON_ARCADEP_MENU
DIBUTTON_ARCADEP_SELECT
DIBUTTON_ARCADEP_SPECIAL

Priority 2 Controls

DIBUTTON_ARCADEP_BACK_LINK
DIBUTTON_ARCADEP_DEVICE
DIBUTTON_ARCADEP_FIRESECONDARY
DIBUTTON_ARCADEP_FORWARD_LINK
DIBUTTON_ARCADEP_LEFT_LINK
DIBUTTON_ARCADEP_PAUSE
DIBUTTON_ARCADEP_RIGHT_LINK
DIBUTTON_ARCADEP_VIEW_DOWN_LINK
DIBUTTON_ARCADEP_VIEW_LEFT_LINK
DIBUTTON_ARCADEP_VIEW_RIGHT_LINK
DIBUTTON_ARCADEP_VIEW_UP_LINK
DIHATSWITCH_ARCADEP_VIEW

Side-to-Side

The DIVIRTUAL_ARCADE_SIDE2SIDE genre contains controls for a two-dimensional arcade-style game.

Priority 1 Controls

DIAXIS_ARCADES_LATERAL
DIAXIS_ARCADES_MOVE
DIBUTTON_ARCADES_ATTACK
DIBUTTON_ARCADES_CARRY
DIBUTTON_ARCADES_MENU
DIBUTTON_ARCADES_SELECT
DIBUTTON_ARCADES_SPECIAL
DIBUTTON_ARCADES_THROW

Priority 2 Controls

DIBUTTON_ARCADES_BACK_LINK
DIBUTTON_ARCADES_DEVICE
DIBUTTON_ARCADES_FORWARD_LINK
DIBUTTON_ARCADES_LEFT_LINK
DIBUTTON_ARCADES_PAUSE
DIBUTTON_ARCADES_RIGHT_LINK
DIBUTTON_ARCADES_VIEW_DOWN_LINK
DIBUTTON_ARCADES_VIEW_LEFT_LINK
DIBUTTON_ARCADES_VIEW_RIGHT_LINK
DIBUTTON_ARCADES_VIEW_UP_LINK
DIHATSWITCH_ARCADES_VIEW

CAD Genres

The following genres contain controls for computer-assisted drafting applications.

- 2-D Object
- 3-D Model
- 3-D Navigation
- 3-D Object

2-D Object

The DIVIRTUAL_CAD_2DCONTROL genre contains controls to select and move objects in a two-dimensional drafting environment.

Priority 1 Controls

DIAXIS_2DCONTROL_LATERAL
DIAXIS_2DCONTROL_MOVE
DIAXIS_2DCONTROL_INOUT
DIBUTTON_2DCONTROL_MENU
DIBUTTON_2DCONTROL_SELECT
DIBUTTON_2DCONTROL_SPECIAL
DIBUTTON_2DCONTROL_SPECIAL1
DIBUTTON_2DCONTROL_SPECIAL2

Priority 2 Controls

DIAXIS_2DCONTROL_ROTATEZ
DIBUTTON_2DCONTROL_DEVICE
DIBUTTON_2DCONTROL_DISPLAY

DIBUTTON_2DCONTROL_PAUSE
DIHATSWITCH_2DCONTROL_HATSWITCH

3-D Model

The DIVIRTUAL_CAD_MODEL genre contains controls for modeling three-dimensional objects.

Priority 1 Controls
DIAXIS_CADM_LATERAL
DIAXIS_CADM_MOVE
DIAXIS_CADM_INOUT
DIBUTTON_CADM_MENU
DIBUTTON_CADM_SELECT
DIBUTTON_CADM_SPECIAL
DIBUTTON_CADM_SPECIAL1
DIBUTTON_CADM_SPECIAL2

Priority 2 Controls
DIAXIS_CADM_ROTATEX
DIAXIS_CADM_ROTATEY
DIAXIS_CADM_ROTATEZ
DIBUTTON_CADM_DEVICE
DIBUTTON_CADM_DISPLAY
DIBUTTON_CADM_PAUSE
DIHATSWITCH_CADM_HATSWITCH

3-D Navigation

The DIVIRTUAL_CAD_FLYBY genre contains control for fly-through navigation of three-dimensional environments.

Priority 1 Controls
DIAXIS_CADF_LATERAL
DIAXIS_CADF_MOVE
DIAXIS_CADF_INOUT
DIBUTTON_CADF_MENU
DIBUTTON_CADF_SELECT
DIBUTTON_CADF_SPECIAL
DIBUTTON_CADF_SPECIAL1
DIBUTTON_CADF_SPECIAL2

Priority 2 Controls
DIAXIS_CADF_ROTATEX
DIAXIS_CADF_ROTATEY
DIAXIS_CADF_ROTATEZ
DIBUTTON_CADF_DEVICE
DIBUTTON_CADF_DISPLAY
DIBUTTON_CADF_PAUSE
DIHATSWITCH_CADF_HATSWITCH

3-D Object

The DIVIRTUAL_CAD_3DCONTROL genre contains controls to select and move objects in a three-dimensional drafting environment.

Priority 1 Controls
DIAXIS_3DCONTROL_LATERAL
DIAXIS_3DCONTROL_MOVE
DIAXIS_3DCONTROL_INOUT
DIBUTTON_3DCONTROL_MENU
DIBUTTON_3DCONTROL_SELECT
DIBUTTON_3DCONTROL_SPECIAL
DIBUTTON_3DCONTROL_SPECIAL1
DIBUTTON_3DCONTROL_SPECIAL2

Priority 2 Controls
DIAXIS_3DCONTROL_ROTATEX
DIAXIS_3DCONTROL_ROTATEY
DIAXIS_3DCONTROL_ROTATEZ
DIBUTTON_3DCONTROL_DEVICE
DIBUTTON_3DCONTROL_DISPLAY
DIBUTTON_3DCONTROL_PAUSE
DIHATSWITCH_3DCONTROL_HATSWITCH

Control Genres

The following genres contain controls for use with Web browsers and remote-control devices.

- Browser
- Remote Control

Browser

The DIVIRTUAL_BROWSER_CONTROL genre contains controls for a Web browser.

Priority 1 Controls

DIAXIS_BROWSER_LATERAL
DIAXIS_BROWSER_MOVE
DIAXIS_BROWSER_VIEW
DIBUTTON_BROWSER_MENU
DIBUTTON_BROWSER_REFRESH
DIBUTTON_BROWSER_SELECT

Priority 2 Controls

DIBUTTON_BROWSER_DEVICE
DIBUTTON_BROWSER_FAVORITES
DIBUTTON_BROWSER_HISTORY
DIBUTTON_BROWSER_HOME
DIBUTTON_BROWSER_NEXT
DIBUTTON_BROWSER_PAUSE
DIBUTTON_BROWSER_PREVIOUS
DIBUTTON_BROWSER_PRINT
DIBUTTON_BROWSER_SEARCH
DIBUTTON_BROWSER_STOP

Remote Control

The DIVIRTUAL_REMOTE_CONTROL genre contains controls for remote-control devices used with equipment such as media players.

Priority 1 Controls

DIAXIS_REMOTE_SLIDER
DIBUTTON_REMOTE_CHANGE
DIBUTTON_REMOTE_CUE
DIBUTTON_REMOTE_MENU
DIBUTTON_REMOTE_MUTE
DIBUTTON_REMOTE_PLAY
DIBUTTON_REMOTE_RECORD
DIBUTTON_REMOTE_REVIEW
DIBUTTON_REMOTE_SELECT

Priority 2 Controls

DIAXIS_REMOTE_SLIDER2
DIBUTTON_REMOTE_ADJUST
DIBUTTON_REMOTE_CABLE
DIBUTTON_REMOTE_CD
DIBUTTON_REMOTE_DEVICE
DIBUTTON_REMOTE_DIGIT0
DIBUTTON_REMOTE_DIGIT1
DIBUTTON_REMOTE_DIGIT2
DIBUTTON_REMOTE_DIGIT3
DIBUTTON_REMOTE_DIGIT4
DIBUTTON_REMOTE_DIGIT5
DIBUTTON_REMOTE_DIGIT6
DIBUTTON_REMOTE_DIGIT7
DIBUTTON_REMOTE_DIGIT8
DIBUTTON_REMOTE_DIGIT9
DIBUTTON_REMOTE_DVD
DIBUTTON_REMOTE_PAUSE
DIBUTTON_REMOTE_TUNER
DIBUTTON_REMOTE_TV
DIBUTTON_REMOTE_VCR

Driving Genres

The following genres are used in games where the players control vehicles in racing or combat.

- Combat Racing
- Mechanical Fighting
- Racing
- Tank

Combat Racing

The DIVIRTUAL_DRIVING_COMBAT genre contains controls for a vehicle race game with combat.

Priority 1 Controls

DIAXIS_DRIVINGC_ACCELERATE
DIAXIS_DRIVINGC_BRAKE

DIAXIS_DRIVINGC_STEER
DIBUTTON_DRIVINGC_FIRE
DIBUTTON_DRIVINGC_MENU
DIBUTTON_DRIVINGC_TARGET
DIBUTTON_DRIVINGC_WEAPONS

Priority 2 Controls
DIBUTTON_DRIVINGC_ACCELERATE-LINK
DIBUTTON_DRIVINGC_AIDS
DIBUTTON_DRIVINGC_BRAKE
DIBUTTON_DRIVINGC_DASHBOARD
DIBUTTON_DRIVINGC_DEVICE
DIBUTTON_DRIVINGC_FIRESECONDARY
DIBUTTON_DRIVINGC_GLANCE_LEFT_LINK
DIBUTTON_DRIVINGC_GLANCE_RIGHT_LINK
DIBUTTON_DRIVINGC_PAUSE
DIBUTTON_DRIVINGC_SHIFTDOWN
DIBUTTON_DRIVINGC_SHIFTUP
DIBUTTON_DRIVINGC_STEER_LEFT_LINK
DIBUTTON_DRIVINGC_STEER_RIGHT_LINK
DIHATSWITCH_DRIVINGC_GLANCE

Mechanical Fighting

The DIVIRTUAL_DRIVING_MECHA genre contains controls for a walking vehicle in a mechanized-warfare game.

Priority 1 Controls
DIAXIS_MECHA_ROTATE
DIAXIS_MECHA_STEER
DIAXIS_MECHA_THROTTLE
DIAXIS_MECHA_TORSO
DIBUTTON_MECHA_FIRE
DIBUTTON_MECHA_JUMP
DIBUTTON_MECHA_MENU
DIBUTTON_MECHA_REVERSE
DIBUTTON_MECHA_TARGET
DIBUTTON_MECHA_WEAPONS
DIBUTTON_MECHA_ZOOM

Priority 2 Controls

DIBUTTON_MECHA_BACK_LINK
DIBUTTON_MECHA_CENTER
DIBUTTON_MECHA_DEVICE
DIBUTTON_MECHA_FASTER_LINK
DIBUTTON_MECHA_FIRESECONDARY
DIBUTTON_MECHA_FORWARD_LINK
DIBUTTON_MECHA_LEFT_LINK
DIBUTTON_MECHA_PAUSE
DIBUTTON_MECHA_RIGHT_LINK
DIBUTTON_MECHA_ROTATE_LEFT_LINK
DIBUTTON_MECHA_ROTATE_RIGHT_LINK
DIBUTTON_MECHA_SLOWER_LINK
DIBUTTON_MECHA_VIEW
DIHATSWITCH_MECHA_GLANCE

Racing

The DIVIRTUAL_DRIVING_RACE genre contains controls for an auto-racing game without combat.

Priority 1 Controls

DIAXIS_DRIVINGR_ACCELERATE
DIAXIS_DRIVINGR_BRAKE
DIAXIS_DRIVINGR_STEER
DIBUTTON_DRIVINGR_MENU
DIBUTTON_DRIVINGR_SHIFTDOWN
DIBUTTON_DRIVINGR_SHIFTUP
DIBUTTON_DRIVINGR_VIEW

Priority 2 Controls

DIAXIS_DRIVINGR_ACCEL_AND_BRAKE
DIBUTTON_DRIVINGR_ACCELERATE_LINK
DIBUTTON_DRIVINGR_AIDS
DIBUTTON_DRIVINGR_BOOST
DIBUTTON_DRIVINGR_BRAKE
DIBUTTON_DRIVINGR_DASHBOARD
DIBUTTON_DRIVINGR_DEVICE
DIBUTTON_DRIVINGR_GLANCE_LEFT_LINK
DIBUTTON_DRIVINGR_GLANCE_RIGHT_LINK

DIBUTTON_DRIVINGR_MAP
DIBUTTON_DRIVINGR_PAUSE
DIBUTTON_DRIVINGR_PIT
DIBUTTON_DRIVINGR_STEER_LEFT_LINK
DIBUTTON_DRIVINGR_STEER_RIGHT_LINK
DIHATSWITCH_DRIVINGR_GLANCE

Tank

The DIVIRTUAL_DRIVING_TANK genre contains controls for a military tank simulation.

Priority 1 Controls

DIAXIS_DRIVINGT_ACCELERATE
DIAXIS_DRIVINGT_BARREL
DIAXIS_DRIVINGT_STEER
DIBUTTON_DRIVINGT_FIRE
DIBUTTON_DRIVINGT_MENU
DIBUTTON_DRIVINGT_TARGET
DIBUTTON_DRIVINGT_WEAPONS

Priority 2 Controls

DIAXIS_DRIVINGT_ACCEL_AND_BRAKE
DIAXIS_DRIVINGT_BRAKE
DIAXIS_DRIVINGT_ACCELERATE_LINK
DIAXIS_DRIVINGT_STEER_LEFT_LINK
DIAXIS_DRIVINGT_STEER_RIGHT_LINK
DIAXIS_DRIVINGT_BARREL_UP_LINK
DIAXIS_DRIVINGT_BARREL_DOWN_LINK
DIAXIS_DRIVINGT_GLANCE_LEFT_LINK
DIAXIS_DRIVINGT_GLANCE_RIGHT_LINK
DIBUTTON_DRIVINGT_BRAKE
DIBUTTON_DRIVINGT_DASHBOARD
DIBUTTON_DRIVINGT_DEVICE
DIBUTTON_DRIVINGT_FIRESECONDARY
DIBUTTON_DRIVINGT_PAUSE
DIBUTTON_DRIVINGT_VIEW
DIHATSWITCH_DRIVINGT_GLANCE

Flight Genres

The following genres are used in games and simulations where the players control aircraft or spaceships.

- Air Combat
- Civilian Flight
- Helicopter Combat
- Space Combat

Air Combat

The DIVIRTUAL_FLYING_MILITARY genre contains controls for a fixed-wing aerial combat simulation.

Priority 1 Controls

DIAXIS_FLYINGM_BANK
DIAXIS_FLYINGM_PITCH
DIAXIS_FLYINGM_THROTTLE
DIBUTTON_FLYINGM_FIRE
DIBUTTON_FLYINGM_MENU
DIBUTTON_FLYINGM_TARGET
DIBUTTON_FLYINGM_WEAPONS

Priority 2 Controls

DIAXIS_FLYINGM_BRAKE
DIAXIS_FLYINGM_FLAPS
DIAXIS_FLYINGM_RUDDER
DIBUTTON_FLYINGM_BRAKE_LINK
DIBUTTON_FLYINGM_COUNTER
DIBUTTON_FLYINGM_DEVICE
DIBUTTON_FLYINGM_DISPLAY
DIBUTTON_FLYINGM_FASTER_LINK
DIBUTTON_FLYINGM_FIRESECONDARY
DIBUTTON_FLYINGM_FLAPSDOWN
DIBUTTON_FLYINGM_FLAPSUP
DIBUTTON_FLYINGM_GEAR
DIBUTTON_FLYINGM_GLANCE_DOWN_LINK
DIBUTTON_FLYINGM_GLANCE_LEFT_LINK
DIBUTTON_FLYINGM_GLANCE_RIGHT_LINK
DIBUTTON_FLYINGM_GLANCE_UP_LINK

DIBUTTON_FLYINGM_PAUSE
DIBUTTON_FLYINGM_SLOWER_LINK
DIBUTTON_FLYINGM_VIEW
DIHATSWITCH_FLYINGM_GLANCE

Civilian Flight

The DIVIRTUAL_FLYING_CIVILIAN genre contains controls for noncombat flight simulations.

Priority 1 Controls
DIAXIS_FLYINGC_BANK
DIAXIS_FLYINGC_PITCH
DIAXIS_FLYINGC_THROTTLE
DIBUTTON_FLYINGC_DISPLAY
DIBUTTON_FLYINGC_GEAR
DIBUTTON_FLYINGC_MENU
DIBUTTON_FLYINGC_VIEW

Priority 2 Controls
DIAXIS_FLYINGC_BRAKE
DIAXIS_FLYINGC_FLAPS
DIAXIS_FLYINGC_RUDDER
DIBUTTON_FLYINGC_BRAKE_LINK
DIBUTTON_FLYINGC_DEVICE
DIBUTTON_FLYINGC_FASTER_LINK
DIBUTTON_FLYINGC_FLAPSDOWN
DIBUTTON_FLYINGC_FLAPSUP
DIBUTTON_FLYINGC_GLANCE_DOWN_LINK
DIBUTTON_FLYINGC_GLANCE_LEFT_LINK
DIBUTTON_FLYINGC_GLANCE_RIGHT_LINK
DIBUTTON_FLYINGC_GLANCE_UP_LINK
DIBUTTON_FLYINGC_PAUSE
DIBUTTON_FLYINGC_SLOWER_LINK
DIHATSWITCH_FLYINGC_GLANCE

Helicopter Combat

The DIVIRTUAL_FLYING_HELICOPTER genre contains controls for a combat helicopter simulation.

Priority 1 Controls

DIAXIS_FLYINGH_BANK
DIAXIS_FLYINGH_COLLECTIVE
DIAXIS_FLYINGH_PITCH
DIBUTTON_FLYINGH_FIRE
DIBUTTON_FLYINGH_MENU
DIBUTTON_FLYINGH_TARGET
DIBUTTON_FLYINGH_WEAPONS

Priority 2 Controls

DIAXIS_FLYINGH_THROTTLE
DIAXIS_FLYINGH_TORQUE
DIBUTTON_FLYINGH_COUNTER
DIBUTTON_FLYINGH_DEVICE
DIBUTTON_FLYINGH_FASTER_LINK
DIBUTTON_FLYINGH_FIRESECONDARY
DIBUTTON_FLYINGH_GEAR
DIBUTTON_FLYINGH_GLANCE_DOWN_LINK
DIBUTTON_FLYINGH_GLANCE_LEFT_LINK
DIBUTTON_FLYINGH_GLANCE_RIGHT_LINK
DIBUTTON_FLYINGH_GLANCE_UP_LINK
DIBUTTON_FLYINGH_PAUSE
DIBUTTON_FLYINGH_SLOWER_LINK
DIBUTTON_FLYINGH_VIEW
DIHATSWITCH_FLYINGH_GLANCE

Space Combat

The DIVIRTUAL_SPACESIM genre contains controls for a spaceship combat simulation.

Priority 1 Controls

DIAXIS_SPACESIM_LATERAL
DIAXIS_SPACESIM_MOVE
DIAXIS_SPACESIM_THROTTLE
DIBUTTON_SPACESIM_FIRE

DIBUTTON_SPACESIM_MENU
DIBUTTON_SPACESIM_TARGET
DIBUTTON_SPACESIM_WEAPONS

Priority 2 Controls
DIAXIS_SPACESIM_CLIMB
DIAXIS_SPACESIM_ROTATE
DIBUTTON_SPACESIM_BACKWARD_LINK
DIBUTTON_SPACESIM_DEVICE
DIBUTTON_SPACESIM_DISPLAY
DIBUTTON_SPACESIM_FASTER_LINK
DIBUTTON_SPACESIM_FIRESECONDARY
DIBUTTON_SPACESIM_FORWARD_LINK
DIBUTTON_SPACESIM_GEAR
DIBUTTON_SPACESIM_GLANCE_DOWN_LINK
DIBUTTON_SPACESIM_GLANCE_LEFT_LINK
DIBUTTON_SPACESIM_GLANCE_RIGHT_LINK
DIBUTTON_SPACESIM_GLANCE_UP_LINK
DIBUTTON_SPACESIM_LEFT_LINK
DIBUTTON_SPACESIM_LOWER
DIBUTTON_SPACESIM_PAUSE
DIBUTTON_SPACESIM_RAISE
DIBUTTON_SPACESIM_RIGHT_LINK
DIBUTTON_SPACESIM_SLOWER_LINK
DIBUTTON_SPACESIM_TURN_LEFT_LINK
DIBUTTON_SPACESIM_TURN_RIGHT_LINK
DIBUTTON_SPACESIM_VIEW
DIHATSWITCH_SPACESIM_GLANCE

Sports Genres

The following genres contain controls for sports games and simulations.

- Baseball Batting
- Baseball Fielding
- Baseball Pitching
- Basketball Defense
- Basketball Offense
- Fishing

- Football Defense
- Football Offense
- Football Play
- Football Quarterback
- Golf
- Hockey Defense
- Hockey Goalie
- Hockey Offense
- Hunting
- Mountain Biking
- Racquet
- Skiing
- Soccer Defense
- Soccer Offense

Baseball Batting

The DIVIRTUAL_SPORTS_BASEBALL_BAT genre contains controls for a batter in a baseball game.

Priority 1 Controls

DIAXIS_BASEBALLB_LATERAL
DIAXIS_BASEBALLB_MOVE
DIBUTTON_BASEBALLB_BUNT
DIBUTTON_BASEBALLB_BURST
DIBUTTON_BASEBALLB_CONTACT
DIBUTTON_BASEBALLB_MENU
DIBUTTON_BASEBALLB_NORMAL
DIBUTTON_BASEBALLB_POWER
DIBUTTON_BASEBALLB_SELECT
DIBUTTON_BASEBALLB_SLIDE
DIBUTTON_BASEBALLB_STEAL

Priority 2 Controls

DIBUTTON_BASEBALLB_BACK_LINK
DIBUTTON_BASEBALLB_BOX
DIBUTTON_BASEBALLB_DEVICE
DIBUTTON_BASEBALLB_FORWARD_LINK
DIBUTTON_BASEBALLB_LEFT_LINK

DIBUTTON_BASEBALLB_NOSTEAL
DIBUTTON_BASEBALLB_PAUSE
DIBUTTON_BASEBALLB_RIGHT_LINK

Baseball Fielding

The DIVIRTUAL_SPORTS_BASEBALL_FIELD genre contains controls for fielders in a baseball game.

Priority 1 Controls

DIAXIS_BASEBALLF_LATERAL
DIAXIS_BASEBALLF_MOVE
DIBUTTON_BASEBALLF_BURST
DIBUTTON_BASEBALLF_DIVE
DIBUTTON_BASEBALLF_JUMP
DIBUTTON_BASEBALLF_MENU
DIBUTTON_BASEBALLF_NEAREST
DIBUTTON_BASEBALLF_THROW1
DIBUTTON_BASEBALLF_THROW2

Priority 2 Controls

DIBUTTON_BASEBALLF_AIM_LEFT_LINK
DIBUTTON_BASEBALLF_AIM_RIGHT_LINK
DIBUTTON_BASEBALLF_BACK_LINK
DIBUTTON_BASEBALLF_DEVICE
DIBUTTON_BASEBALLF_FORWARD_LINK
DIBUTTON_BASEBALLF_PAUSE
DIBUTTON_BASEBALLF_SHIFTIN
DIBUTTON_BASEBALLF_SHIFTOUT

Baseball Pitching

The DIVIRTUAL_SPORTS_BASEBALL_PITCH genre contains controls for a pitcher in a baseball game.

Priority 1 Controls

DIAXIS_BASEBALLP_LATERAL
DIAXIS_BASEBALLP_MOVE
DIBUTTON_BASEBALLP_BASE
DIBUTTON_BASEBALLP_FAKE
DIBUTTON_BASEBALLP_MENU

DIBUTTON_BASEBALLP_PITCH
DIBUTTON_BASEBALLP_SELECT
DIBUTTON_BASEBALLP_THROW

Priority 2 Controls
DIBUTTON_BASEBALLP_BACK_LINK
DIBUTTON_BASEBALLP_DEVICE
DIBUTTON_BASEBALLP_FORWARD_LINK
DIBUTTON_BASEBALLP_LEFT_LINK
DIBUTTON_BASEBALLP_LOOK
DIBUTTON_BASEBALLP_PAUSE
DIBUTTON_BASEBALLP_RIGHT_LINK
DIBUTTON_BASEBALLP_WALK

Basketball Defense

The DIVIRTUAL_SPORTS_BASKETBALL_DEFENSE genre contains controls for defensive moves in a basketball game.

Priority 1 Controls
DIAXIS_BBALLD_LATERAL
DIAXIS_BBALLD_MOVE
DIBUTTON_BBALLD_BURST
DIBUTTON_BBALLD_FAKE
DIBUTTON_BBALLD_JUMP
DIBUTTON_BBALLD_MENU
DIBUTTON_BBALLD_PLAY
DIBUTTON_BBALLD_PLAYER
DIBUTTON_BBALLD_SPECIAL
DIBUTTON_BBALLD_STEAL

Priority 2 Controls
DIBUTTON_BBALLD_BACK_LINK
DIBUTTON_BBALLD_DEVICE
DIBUTTON_BBALLD_FORWARD_LINK
DIBUTTON_BBALLD_LEFT_LINK
DIBUTTON_BBALLD_PAUSE
DIBUTTON_BBALLD_RIGHT_LINK
DIBUTTON_BBALLD_SUBSTITUTE
DIBUTTON_BBALLD_TIMEOUT

DIHATSWITCH_BBALLD_GLANCE

Basketball Offense

The DIVIRTUAL_SPORTS_BASKETBALL_OFFENSE genre contains controls for the player who has the ball in a basketball game.

Priority 1 Controls

DIAXIS_BBALLO_LATERAL
DIAXIS_BBALLO_MOVE
DIBUTTON_BBALLO_BURST
DIBUTTON_BBALLO_CALL
DIBUTTON_BBALLO_DUNK
DIBUTTON_BBALLO_FAKE
DIBUTTON_BBALLO_MENU
DIBUTTON_BBALLO_PASS
DIBUTTON_BBALLO_PLAYER
DIBUTTON_BBALLO_SHOOT
DIBUTTON_BBALLO_SPECIAL

Priority 2 Controls

DIBUTTON_BBALLO_BACK_LINK
DIBUTTON_BBALLO_DEVICE
DIBUTTON_BBALLO_FORWARD_LINK
DIBUTTON_BBALLO_JAB
DIBUTTON_BBALLO_LEFT_LINK
DIBUTTON_BBALLO_PAUSE
DIBUTTON_BBALLO_PLAY
DIBUTTON_BBALLO_POST
DIBUTTON_BBALLO_RIGHT_LINK
DIBUTTON_BBALLO_SCREEN
DIBUTTON_BBALLO_SUBSTITUTE
DIBUTTON_BBALLO_TIMEOUT
DIHATSWITCH_BBALLO_GLANCE

Fishing

The DIVIRTUAL_SPORTS_FISHING genre contains controls for a sport-fishing simulation.

Priority 1 Controls

DIAXIS_FISHING_LATERAL
DIAXIS_FISHING_MOVE
DIBUTTON_FISHING_BAIT
DIBUTTON_FISHING_BINOCULAR
DIBUTTON_FISHING_CAST
DIBUTTON_FISHING_MAP
DIBUTTON_FISHING_MENU
DIBUTTON_FISHING_TYPE

Priority 2 Controls
DIAXIS_FISHING_ROTATE
DIBUTTON_FISHING_BACK_LINK
DIBUTTON_FISHING_CROUCH
DIBUTTON_FISHING_DEVICE
DIBUTTON_FISHING_DISPLAY
DIBUTTON_FISHING_FORWARD_LINK
DIBUTTON_FISHING_JUMP
DIBUTTON_FISHING_LEFT_LINK
DIBUTTON_FISHING_PAUSE
DIBUTTON_FISHING_RIGHT_LINK
DIBUTTON_FISHING_ROTATE_LEFT_LINK
DIBUTTON_FISHING_ROTATE_RIGHT_LINK
DIHATSWITCH_FISHING_GLANCE

Football Defense

The DIVIRTUAL_SPORTS_FOOTBALL_DEFENSE genre contains controls for the defensive side in a North American football game.

Priority 1 Controls
DIAXIS_FOOTBALLD_LATERAL
DIAXIS_FOOTBALLD_MOVE
DIBUTTON_FOOTBALLD_FAKE
DIBUTTON_FOOTBALLD_JUMP
DIBUTTON_FOOTBALLD_MENU
DIBUTTON_FOOTBALLD_PLAY
DIBUTTON_FOOTBALLD_SELECT
DIBUTTON_FOOTBALLD_SUPERTACKLE
DIBUTTON_FOOTBALLD_TACKLE

Priority 2 Controls
DIBUTTON_FOOTBALLD_AUDIBLE
DIBUTTON_FOOTBALLD_BACK_LINK
DIBUTTON_FOOTBALLD_BULLRUSH
DIBUTTON_FOOTBALLD_DEVICE
DIBUTTON_FOOTBALLD_FORWARD_LINK
DIBUTTON_FOOTBALLD_LEFT_LINK
DIBUTTON_FOOTBALLD_PAUSE
DIBUTTON_FOOTBALLD_RIGHT_LINK
DIBUTTON_FOOTBALLD_RIP
DIBUTTON_FOOTBALLD_SPIN
DIBUTTON_FOOTBALLD_SUBSTITUTE
DIBUTTON_FOOTBALLD_SWIM
DIBUTTON_FOOTBALLD_ZOOM

Football Offense

The DIVIRTUAL_SPORTS_FOOTBALL_OFFENSE genre contains controls for the ball carrier in a North American football game.

Priority 1 Controls
DIAXIS_FOOTBALLO_LATERAL
DIAXIS_FOOTBALLO_MOVE
DIBUTTON_FOOTBALLO_JUMP
DIBUTTON_FOOTBALLO_LEFTARM
DIBUTTON_FOOTBALLO_MENU
DIBUTTON_FOOTBALLO_RIGHTARM
DIBUTTON_FOOTBALLO_SPIN
DIBUTTON_FOOTBALLO_THROW

Priority 2 Controls
DIBUTTON_FOOTBALLO_BACK_LINK
DIBUTTON_FOOTBALLO_DEVICE
DIBUTTON_FOOTBALLO_DIVE
DIBUTTON_FOOTBALLO_FORWARD_LINK
DIBUTTON_FOOTBALLO_JUKE
DIBUTTON_FOOTBALLO_LEFT_LINK
DIBUTTON_FOOTBALLO_PAUSE
DIBUTTON_FOOTBALLO_RIGHT_LINK
DIBUTTON_FOOTBALLO_SHOULDER

DIBUTTON_FOOTBALLO_SUBSTITUTE
DIBUTTON_FOOTBALLO_TURBO
DIBUTTON_FOOTBALLO_ZOOM

Football Play

The DIVIRTUAL_SPORTS_FOOTBALL_FIELD genre contains general controls for a North American football game.

Priority 1 Controls
DIBUTTON_FOOTBALLP_HELP
DIBUTTON_FOOTBALLP_MENU
DIBUTTON_FOOTBALLP_PLAY
DIBUTTON_FOOTBALLP_SELECT

Priority 2 Controls
DIBUTTON_FOOTBALLP_DEVICE
DIBUTTON_FOOTBALLP_PAUSE

Football Quarterback

The DIVIRTUAL_SPORTS_FOOTBALL_QBCK genre contains controls for the quarterback in a North American football game.

Priority 1 Controls
DIAXIS_FOOTBALLQ_LATERAL
DIAXIS_FOOTBALLQ_MOVE
DIBUTTON_FOOTBALLQ_FAKE
DIBUTTON_FOOTBALLQ_JUMP
DIBUTTON_FOOTBALLQ_MENU
DIBUTTON_FOOTBALLQ_PASS
DIBUTTON_FOOTBALLQ_SELECT
DIBUTTON_FOOTBALLQ_SLIDE
DIBUTTON_FOOTBALLQ_SNAP

Priority 2 Controls
DIBUTTON_FOOTBALLQ_AUDIBLE
DIBUTTON_FOOTBALLQ_BACK_LINK
DIBUTTON_FOOTBALLQ_DEVICE
DIBUTTON_FOOTBALLQ_FAKESNAP

DIBUTTON_FOOTBALLQ_FORWARD_LINK
DIBUTTON_FOOTBALLQ_LEFT_LINK
DIBUTTON_FOOTBALLQ_MOTION
DIBUTTON_FOOTBALLQ_PAUSE
DIBUTTON_FOOTBALLQ_RIGHT_LINK

Golf

The DIVIRTUAL_SPORTS_GOLF genre contains controls for a golf game.

Priority 1 Controls
DIAXIS_GOLF_LATERAL
DIAXIS_GOLF_MOVE
DIBUTTON_GOLF_DOWN
DIBUTTON_GOLF_FLYBY
DIBUTTON_GOLF_MENU
DIBUTTON_GOLF_SELECT
DIBUTTON_GOLF_SWING
DIBUTTON_GOLF_TERRAIN
DIBUTTON_GOLF_UP

Priority 2 Controls
DIBUTTON_GOLF_BACK_LINK
DIBUTTON_GOLF_DEVICE
DIBUTTON_GOLF_FORWARD_LINK
DIBUTTON_GOLF_LEFT_LINK
DIBUTTON_GOLF_PAUSE
DIBUTTON_GOLF_RIGHT_LINK
DIBUTTON_GOLF_SUBSTITUTE
DIBUTTON_GOLF_TIMEOUT
DIBUTTON_GOLF_ZOOM
DIHATSWITCH_GOLF_SCROLL

Hockey Defense

The DIVIRTUAL_SPORTS_HOCKEY_DEFENSE genre contains controls for defensive moves in a hockey game.

Priority 1 Controls
DIAXIS_HOCKEYD_LATERAL
DIAXIS_HOCKEYD_MOVE

DIBUTTON_HOCKEYD_BLOCK
DIBUTTON_HOCKEYD_BURST
DIBUTTON_HOCKEYD_FAKE
DIBUTTON_HOCKEYD_MENU
DIBUTTON_HOCKEYD_PLAYER
DIBUTTON_HOCKEYD_STEAL

Priority 2 Controls
DIBUTTON_HOCKEYD_BACK_LINK
DIBUTTON_HOCKEYD_DEVICE
DIBUTTON_HOCKEYD_FORWARD_LINK
DIBUTTON_HOCKEYD_LEFT_LINK
DIBUTTON_HOCKEYD_PAUSE
DIBUTTON_HOCKEYD_RIGHT_LINK
DIBUTTON_HOCKEYD_STRATEGY
DIBUTTON_HOCKEYD_SUBSTITUTE
DIBUTTON_HOCKEYD_TIMEOUT
DIBUTTON_HOCKEYD_ZOOM
DIHATSWITCH_HOCKEYD_SCROLL

Hockey Goalie

The DIVIRTUAL_SPORTS_HOCKEY_GOALIE genre contains controls for a goalkeeper in a hockey game.

Priority 1 Controls
DIAXIS_HOCKEYG_LATERAL
DIAXIS_HOCKEYG_MOVE
DIBUTTON_HOCKEYG_BLOCK
DIBUTTON_HOCKEYG_MENU
DIBUTTON_HOCKEYG_PASS
DIBUTTON_HOCKEYG_POKE
DIBUTTON_HOCKEYG_STEAL

Priority 2 Controls
DIBUTTON_HOCKEYG_BACK_LINK
DIBUTTON_HOCKEYG_DEVICE
DIBUTTON_HOCKEYG_FORWARD_LINK
DIBUTTON_HOCKEYG_LEFT_LINK

DIBUTTON_HOCKEYG_PAUSE
DIBUTTON_HOCKEYG_RIGHT_LINK
DIBUTTON_HOCKEYG_STRATEGY
DIBUTTON_HOCKEYG_SUBSTITUTE
DIBUTTON_HOCKEYG_TIMEOUT
DIBUTTON_HOCKEYG_ZOOM
DIHATSWITCH_HOCKEYG_SCROLL

Hockey Offense

The DIVIRTUAL_SPORTS_HOCKEY_OFFENSE genre contains controls for offensive moves in a hockey game.

Priority 1 Controls

DIAXIS_HOCKEYO_LATERAL
DIAXIS_HOCKEYO_MOVE
DIBUTTON_HOCKEYO_BURST
DIBUTTON_HOCKEYO_FAKE
DIBUTTON_HOCKEYO_MENU
DIBUTTON_HOCKEYO_PASS
DIBUTTON_HOCKEYO_SHOOT
DIBUTTON_HOCKEYO_SPECIAL

Priority 2 Controls

DIBUTTON_HOCKEYO_BACK_LINK
DIBUTTON_HOCKEYO_DEVICE
DIBUTTON_HOCKEYO_FORWARD_LINK
DIBUTTON_HOCKEYO_LEFT_LINK
DIBUTTON_HOCKEYO_PAUSE
DIBUTTON_HOCKEYO_RIGHT_LINK
DIBUTTON_HOCKEYO_STRATEGY
DIBUTTON_HOCKEYO_SUBSTITUTE
DIBUTTON_HOCKEYO_TIMEOUT
DIBUTTON_HOCKEYO_ZOOM
DIHATSWITCH_HOCKEYO_SCROLL

Hunting

The DIVIRTUAL_SPORTS_HUNTING genre contains controls for a hunting simulation.

Priority 1 Controls
DIAXIS_HUNTING_LATERAL
DIAXIS_HUNTING_MOVE
DIBUTTON_HUNTING_AIM
DIBUTTON_HUNTING_BINOCULAR
DIBUTTON_HUNTING_CALL
DIBUTTON_HUNTING_FIRE
DIBUTTON_HUNTING_MAP
DIBUTTON_HUNTING_MENU
DIBUTTON_HUNTING_SPECIAL
DIBUTTON_HUNTING_WEAPON

Priority 2 Controls
DIAXIS_HUNTING_ROTATE
DIBUTTON_HUNTING_BACK_LINK
DIBUTTON_HUNTING_CROUCH
DIBUTTON_HUNTING_DEVICE
DIBUTTON_HUNTING_DISPLAY
DIBUTTON_HUNTING_FIRESECONDARY
DIBUTTON_HUNTING_FORWARD_LINK
DIBUTTON_HUNTING_JUMP
DIBUTTON_HUNTING_LEFT_LINK
DIBUTTON_HUNTING_PAUSE
DIBUTTON_HUNTING_RIGHT_LINK
DIBUTTON_HUNTING_ROTATE_LEFT_LINK
DIBUTTON_HUNTING_ROTATE_RIGHT_LINK
DIHATSWITCH_HUNTING_GLANCE

Mountain Biking

The DIVIRTUAL_SPORTS_BIKING_MOUNTAIN contains controls for a mountain-bike game.

Priority 1 Controls
DIAXIS_BIKINGM_PEDAL
DIAXIS_BIKINGM_TURN
DIBUTTON_BIKINGM_CAMERA
DIBUTTON_BIKINGM_JUMP
DIBUTTON_BIKINGM_MENU
DIBUTTON_BIKINGM_SELECT

DIBUTTON_BIKINGM_SPECIAL1
DIBUTTON_BIKINGM_SPECIAL2

Priority 2 Controls
DIAXIS_BIKINGM_BRAKE
DIBUTTON_BIKINGM_BRAKE_BUTTON_LINK
DIBUTTON_BIKINGM_DEVICE
DIBUTTON_BIKINGM_FASTER_LINK
DIBUTTON_BIKINGM_LEFT_LINK
DIBUTTON_BIKINGM_PAUSE
DIBUTTON_BIKINGM_RIGHT_LINK
DIBUTTON_BIKINGM_SLOWER_LINK
DIBUTTON_BIKINGM_ZOOM
DIHATSWITCH_BIKINGM_SCROLL

Racquet

The DIVIRTUAL_SPORTS_RACQUET genre includes racquet games such as tennis, table tennis, and squash.

Priority 1 Controls
DIAXIS_RACQUET_LATERAL
DIAXIS_RACQUET_MOVE
DIBUTTON_RACQUET_BACKSWING
DIBUTTON_RACQUET_MENU
DIBUTTON_RACQUET_SELECT
DIBUTTON_RACQUET_SMASH
DIBUTTON_RACQUET_SPECIAL
DIBUTTON_RACQUET_SWING

Priority 2 Controls
DIBUTTON_RACQUET_BACK_LINK
DIBUTTON_RACQUET_DEVICE
DIBUTTON_RACQUET_FORWARD_LINK
DIBUTTON_RACQUET_LEFT_LINK
DIBUTTON_RACQUET_PAUSE
DIBUTTON_RACQUET_RIGHT_LINK
DIBUTTON_RACQUET_SUBSTITUTE
DIBUTTON_RACQUET_TIMEOUT

DIHATSWITCH_RACQUET_GLANCE

Skiing

The DIVIRTUAL_SPORTS_SKIING contains controls for skiing games as well as for similar sports simulations such as snowboarding and skateboarding.

Priority 1 Controls

DIAXIS_SKIING_SPEED
DIAXIS_SKIING_TURN
DIBUTTON_SKIING_CAMERA
DIBUTTON_SKIING_CROUCH
DIBUTTON_SKIING_JUMP
DIBUTTON_SKIING_MENU
DIBUTTON_SKIING_SELECT
DIBUTTON_SKIING_SPECIAL1
DIBUTTON_SKIING_SPECIAL2

Priority 2 Controls

DIBUTTON_SKIING_DEVICE
DIBUTTON_SKIING_FASTER_LINK
DIBUTTON_SKIING_LEFT_LINK
DIBUTTON_SKIING_PAUSE
DIBUTTON_SKIING_RIGHT_LINK
DIBUTTON_SKIING_SLOWER_LINK
DIBUTTON_SKIING_ZOOM
DIHATSWITCH_SKIING_GLANCE

Soccer Defense

The DIVIRTUAL_SPORTS_SOCCER_DEFENSE contains controls for defensive moves in a soccer (international football) game.

Priority 1 Controls

DIAXIS_SOCCERD_LATERAL
DIAXIS_SOCCERD_MOVE
DIBUTTON_SOCCERD_BLOCK
DIBUTTON_SOCCERD_FAKE
DIBUTTON_SOCCERD_MENU
DIBUTTON_SOCCERD_PLAYER
DIBUTTON_SOCCERD_SELECT

DIBUTTON_SOCCERD_SLIDE
DIBUTTON_SOCCERD_SPECIAL
DIBUTTON_SOCCERD_STEAL

Priority 2 Controls
DIBUTTON_SOCCERD_BACK_LINK
DIBUTTON_SOCCERD_CLEAR
DIBUTTON_SOCCERD_DEVICE
DIBUTTON_SOCCERD_FORWARD_LINK
DIBUTTON_SOCCERD_FOUL
DIBUTTON_SOCCERD_GOALIECHARGE
DIBUTTON_SOCCERD_HEAD
DIBUTTON_SOCCERD_LEFT_LINK
DIBUTTON_SOCCERD_PAUSE
DIBUTTON_SOCCERD_RIGHT_LINK
DIBUTTON_SOCCERD_SUBSTITUTE
DIHATSWITCH_SOCCERD_GLANCE

Soccer Offense

The DIVIRTUAL_SPORTS_SOCCER_OFFENSE contains controls for offensive moves in a soccer (international football) game.

Priority 1 Controls
DIAXIS_SOCCERO_BEND
DIAXIS_SOCCERO_LATERAL
DIAXIS_SOCCERO_MOVE
DIBUTTON_SOCCERO_FAKE
DIBUTTON_SOCCERO_MENU
DIBUTTON_SOCCERO_PASS
DIBUTTON_SOCCERO_PLAYER
DIBUTTON_SOCCERO_SELECT
DIBUTTON_SOCCERO_SHOOT
DIBUTTON_SOCCERO_SPECIAL1

Priority 2 Controls
DIBUTTON_SOCCERD_BACK_LINK
DIBUTTON_SOCCERD_FORWARD_LINK
DIBUTTON_SOCCERD_RIGHT_LINK
DIBUTTON_SOCCERO_CONTROL

DIBUTTON_SOCCERO_DEVICE
DIBUTTON_SOCCERO_HEAD
DIBUTTON_SOCCERO_LEFT_LINK
DIBUTTON_SOCCERO_PASSTHRU
DIBUTTON_SOCCERO_PAUSE
DIBUTTON_SOCCERO_SHOOTHIGH
DIBUTTON_SOCCERO_SHOOTLOW
DIBUTTON_SOCCERO_SPRINT
DIBUTTON_SOCCERO_SUBSTITUTE
DIHATSWITCH_SOCCERO_GLANCE

Strategy Genres

The following genres contain controls for strategy, role-playing, and adventure games.

- Role-Playing
- Turn-Based

Role-Playing

The DIVIRTUAL_STRATEGY_ROLEPLAYING genre contains controls for a role-playing or adventure game in which navigation, problem-solving, and fighting with weapons and magic are common activities.

Priority 1 Controls

DIAXIS_STRATEGYR_LATERAL
DIAXIS_STRATEGYR_MOVE
DIBUTTON_STRATEGYR_APPLY
DIBUTTON_STRATEGYR_ATTACK
DIBUTTON_STRATEGYR_CAST
DIBUTTON_STRATEGYR_CROUCH
DIBUTTON_STRATEGYR_GET
DIBUTTON_STRATEGYR_JUMP
DIBUTTON_STRATEGYR_MENU
DIBUTTON_STRATEGYR_SELECT

Priority 2 Controls

DIAXIS_STRATEGYR_ROTATE
DIBUTTON_STRATEGYR_BACK_LINK
DIBUTTON_STRATEGYR_DEVICE

DIBUTTON_STRATEGYR_DISPLAY
DIBUTTON_STRATEGYR_FORWARD_LINK
DIBUTTON_STRATEGYR_LEFT_LINK
DIBUTTON_STRATEGYR_MAP
DIBUTTON_STRATEGYR_PAUSE
DIBUTTON_STRATEGYR_RIGHT_LINK
DIBUTTON_STRATEGYR_ROTATE_LEFT_LINK
DIBUTTON_STRATEGYR_ROTATE_RIGHT_LINK
DIHATSWITCH_STRATEGYR_GLANCE

Turn-Based

The DIVIRTUAL_STRATEGY_TURN genre contains controls for a turn-based strategy game.

Priority 1 Controls
DIAXIS_STRATEGYT_LATERAL
DIAXIS_STRATEGYT_MOVE
DIBUTTON_STRATEGYT_APPLY
DIBUTTON_STRATEGYT_INSTRUCT
DIBUTTON_STRATEGYT_MENU
DIBUTTON_STRATEGYT_SELECT
DIBUTTON_STRATEGYT_TEAM
DIBUTTON_STRATEGYT_TURN

Priority 2 Controls
DIBUTTON_STRATEGYT_BACK_LIN
DIBUTTON_STRATEGYT_DEVICE
DIBUTTON_STRATEGYT_DISPLAY
DIBUTTON_STRATEGYT_FORWARD_LINK
DIBUTTON_STRATEGYT_LEFT_LINK
DIBUTTON_STRATEGYT_MAP
DIBUTTON_STRATEGYT_PAUSE
DIBUTTON_STRATEGYT_RIGHT_LINK
DIBUTTON_STRATEGYT_ZOOM

Keyboard Mapping Constants

The following constants are used in the **ActionName** member of the **DIACTION** type to map an action to a physical key.

Constant	Note
DIKEYBOARD_0	On main keyboard
DIKEYBOARD_1	On main keyboard
DIKEYBOARD_2	On main keyboard
DIKEYBOARD_3	On main keyboard
DIKEYBOARD_4	On main keyboard
DIKEYBOARD_5	On main keyboard
DIKEYBOARD_6	On main keyboard
DIKEYBOARD_7	On main keyboard
DIKEYBOARD_8	On main keyboard
DIKEYBOARD_9	On main keyboard
DIKEYBOARD_A	
DIKEYBOARD_ABNT_C1	On numeric pad of Brazilian keyboards
DIKEYBOARD_ABNT_C2	On numeric pad of Brazilian keyboards
DIKEYBOARD_ADD	PLUS SIGN (+) on numeric keypad
DIKEYBOARD_APOSTROPHE	
DIKEYBOARD_APPS	
DIKEYBOARD_AT	On Japanese keyboard
DIKEYBOARD_AX	On Japanese keyboard
DIKEYBOARD_B	
DIKEYBOARD_BACK	BACKSPACE
DIKEYBOARD_BACKSLASH	
DIKEYBOARD_C	
DIKEYBOARD_CALCULATOR	
DIKEYBOARD_CAPITAL	CAPS LOCK
DIKEYBOARD_COLON	On Japanese keyboard
DIKEYBOARD_COMMA	
DIKEYBOARD_CONVERT	On Japanese keyboard
DIKEYBOARD_D	
DIKEYBOARD_DECIMAL	PERIOD (decimal point) on numeric keypad
DIKEYBOARD_DELETE	
DIKEYBOARD_DIVIDE	Forward slash (/) on numeric keypad
DIKEYBOARD_DOWN	DOWN ARROW
DIKEYBOARD_E	
DIKEYBOARD_END	
DIKEYBOARD_EQUALS	On main keyboard
DIKEYBOARD_ESCAPE	

DIKEYBOARD_F	
DIKEYBOARD_F1	
DIKEYBOARD_F2	
DIKEYBOARD_F3	
DIKEYBOARD_F4	
DIKEYBOARD_F5	
DIKEYBOARD_F6	
DIKEYBOARD_F7	
DIKEYBOARD_F8	
DIKEYBOARD_F9	
DIKEYBOARD_F10	
DIKEYBOARD_F11	
DIKEYBOARD_F12	
DIKEYBOARD_F13	On NEC PC-98 Japanese keyboard
DIKEYBOARD_F14	On NEC PC-98 Japanese keyboard
DIKEYBOARD_F15	On NEC PC-98 Japanese keyboard
DIKEYBOARD_G	
DIKEYBOARD_GRAVE	Grave accent (`)
DIKEYBOARD_H	
DIKEYBOARD_HOME	
DIKEYBOARD_I	
DIKEYBOARD_INSERT	
DIKEYBOARD_J	
DIKEYBOARD_K	
DIKEYBOARD_KANA	On Japanese keyboard
DIKEYBOARD_KANJI	On Japanese keyboard
DIKEYBOARD_L	
DIKEYBOARD_LBRACKET	Left square bracket [
DIKEYBOARD_LCONTROL	Left CTRL
DIKEYBOARD_LEFT	LEFT ARROW
DIKEYBOARD_LMENU	Left ALT
DIKEYBOARD_LSHIFT	Left SHIFT
DIKEYBOARD_LWIN	Left Microsoft® Windows® logo key
DIKEYBOARD_M	
DIKEYBOARD_MAIL	
DIKEYBOARD_MEDIASELECT	
DIKEYBOARD_MEDIASTOP	
DIKEYBOARD_MINUS	On main keyboard

DIKEYBOARD_MULTIPLY	Asterisk (*) on numeric keypad
DIKEYBOARD_MUTE	
DIKEYBOARD_MYCOMPUTER	
DIKEYBOARD_N	
DIKEYBOARD_NEXT	PAGE DOWN
DIKEYBOARD_NEXTTRACK	Next track
DIKEYBOARD_NOCONVERT	On Japanese keyboard
DIKEYBOARD_NUMLOCK	
DIKEYBOARD_NUMPAD0	
DIKEYBOARD_NUMPAD1	
DIKEYBOARD_NUMPAD2	
DIKEYBOARD_NUMPAD3	
DIKEYBOARD_NUMPAD4	
DIKEYBOARD_NUMPAD5	
DIKEYBOARD_NUMPAD6	
DIKEYBOARD_NUMPAD7	
DIKEYBOARD_NUMPAD8	
DIKEYBOARD_NUMPAD9	
DIKEYBOARD_NUMPADCOMMA	On numeric keypad of NEC PC-98 Japanese keyboard
DIKEYBOARD_NUMPADENTER	
DIKEYBOARD_NUMPADEQUALS	On numeric keypad of NEC PC-98 Japanese keyboard
DIKEYBOARD_O	
DIKEYBOARD_OEM_102	On British and German keyboards
DIKEYBOARD_P	
DIKEYBOARD_PAUSE	
DIKEYBOARD_PERIOD	On main keyboard
DIKEYBOARD_PLAYPAUSE	
DIKEYBOARD_POWER	
DIKEYBOARD_PREVTRACK	Previous track; circumflex on Japanese keyboard
DIKEYBOARD_PRIOR	PAGE UP
DIKEYBOARD_Q	
DIKEYBOARD_R	
DIKEYBOARD_RBRACKET	Right square bracket]
DIKEYBOARD_RCONTROL	Right CTRL
DIKEYBOARD_RETURN	ENTER on main keyboard

DIKEYBOARD_RIGHT	RIGHT ARROW
DIKEYBOARD_RMENU	Right ALT
DIKEYBOARD_RSHIFT	Right SHIFT
DIKEYBOARD_RWIN	Right Windows logo key
DIKEYBOARD_S	
DIKEYBOARD_SCROLL	SCROLL LOCK
DIKEYBOARD_SEMICOLON	
DIKEYBOARD_SLASH	Forward slash (/) on main keyboard
DIKEYBOARD_SLEEP	
DIKEYBOARD_SPACE	SPACEBAR
DIKEYBOARD_STOP	On NEC PC-98 Japanese keyboard
DIKEYBOARD_SUBTRACT	MINUS SIGN (-) on numeric keypad
DIKEYBOARD_SYSRQ	
DIKEYBOARD_T	
DIKEYBOARD_TAB	
DIKEYBOARD_U	
DIKEYBOARD_UNDERLINE	On NEC PC-98 Japanese keyboard
DIKEYBOARD_UNLABELED	On Japanese keyboard
DIKEYBOARD_UP	UP ARROW
DIKEYBOARD_V	
DIKEYBOARD_VOLUMEDOWN	
DIKEYBOARD_VOLUMEUP	
DIKEYBOARD_W	
DIKEYBOARD_WAKE	
DIKEYBOARD_WEBBACK	
DIKEYBOARD_WEBFAVORITES	
DIKEYBOARD_WEBFORWARD	
DIKEYBOARD_WEBHOME	
DIKEYBOARD_WEBREFRESH	
DIKEYBOARD_WEBSEARCH	
DIKEYBOARD_WEBSTOP	
DIKEYBOARD_X	
DIKEYBOARD_Y	
DIKEYBOARD_YEN	On Japanese keyboard
DIKEYBOARD_Z	

Mouse Mapping Constants

The following constants are used in the **ActionName** member of the **DIACTION** type to map an action to a physical axis or button on a mouse.

DIMOUSE_BUTTON0
DIMOUSE_BUTTON1
DIMOUSE_BUTTON2
DIMOUSE_BUTTON3
DIMOUSE_BUTTON4
DIMOUSE_BUTTON5
DIMOUSE_BUTTON6
DIMOUSE_BUTTON7
DIMOUSE_WHEEL
DIMOUSE_XAXIS
DIMOUSE_YAXIS
DIMOUSE_XAXISAB
DIMOUSE_YAXISAB

Note

DIMOUSE_XAXISAB and DIMOUSE_YAXISAB represent the x-axis and y-axis on a mouse that returns absolute rather than relative axis data.

DirectPlay Voice Mapping Constants

The following constants are used in the **ActionName** member of the **DIACTION** type to map an action to a channel on a Microsoft® DirectPlay® voice device.

DIVOICE_CHANNEL1
DIVOICE_CHANNEL2
DIVOICE_CHANNEL3
DIVOICE_CHANNEL4
DIVOICE_CHANNEL5
DIVOICE_CHANNEL6
DIVOICE_CHANNEL7
DIVOICE_CHANNEL8
DIVOICE_ALL
DIVOICE_TEAM
DIVOICE_PLAYBACKMUTE
DIVOICE_RECORDMUTE

DIVOICE_TRANSMIT
DIVOICE_VOICECOMMAND

Any-Control Constants

The following constants are used in the **ActionName** member of the **DIACTION** type to map an action to any matching control on the device.

Constant	Device object
DIAXIS_ANY_1	Any axis
DIAXIS_ANY_2	Any axis
DIAXIS_ANY_3	Any axis
DIAXIS_ANY_4	Any axis
DIAXIS_ANY_A_1	Any accelerator
DIAXIS_ANY_A_2	Any accelerator
DIAXIS_ANY_B_1	Any brake
DIAXIS_ANY_B_2	Any brake
DIAXIS_ANY_C_1	Any clutch
DIAXIS_ANY_C_2	Any clutch
DIAXIS_ANY_R_1	Any r-axis
DIAXIS_ANY_R_2	Any r-axis
DIAXIS_ANY_S_1	Any s-axis
DIAXIS_ANY_S_2	Any s-axis
DIAXIS_ANY_U_1	Any u-axis
DIAXIS_ANY_U_2	Any u-axis
DIAXIS_ANY_V_1	Any v-axis
DIAXIS_ANY_V_2	Any v-axis
DIAXIS_ANY_X_1	Any x-axis
DIAXIS_ANY_X_2	Any x-axis
DIAXIS_ANY_Y_1	Any y-axis
DIAXIS_ANY_Y_2	Any y-axis
DIAXIS_ANY_Z_1	Any z-axis
DIAXIS_ANY_Z_2	Any z-axis
DIBUTTON_ANY	Any button
DIPOV_ANY_1	Any point-of-view controller
DIPOV_ANY_2	Any point-of-view controller
DIPOV_ANY_3	Any point-of-view controller
DIPOV_ANY_4	Any point-of-view controller

These constants can be used to map an application action to a virtual control that is not defined in a genre. Such actions are mapped after genre-specific actions. If the mapper has already mapped all matching controls to genre-specific actions, the any-control action is left unmapped.

Any-control actions are treated with equal priority. If a device has one x-axis and the action array specifies DIAXIS_ANY_1 and DIAXIS_ANY_X_1, the action mapped to the x-axis is the one that appears first in the action array.

Error Codes

This table lists the error codes that can be returned by DirectInput methods and functions. Errors are represented by negative values and cannot be combined.

For a list of the errors each method or function can raise, see the individual descriptions. Lists of error codes in the documentation are necessarily incomplete. For example, any DirectInput method can return DIERR_OUTOFMEMORY even though the error code is not explicitly listed as a possible return value in the documentation for that method.

DI_OK
: The operation completed successfully. This value is equal to the S_OK standard COM return value.

DIERR_ACQUIRED
: The operation cannot be performed while the device is acquired.

DIERR_ALREADYINITIALIZED
: This object is already initialized

DIERR_BADDRIVERVER
: The object could not be created due to an incompatible driver version or mismatched or incomplete driver components.

DIERR_BETADIRECTINPUTVERSION
: The application was written for an unsupported prerelease version of DirectInput.

DIERR_DEVICEFULL
: The device is full.

DIERR_DEVICENOTREG
: The device or device instance is not registered with DirectInput. This value is equal to the REGDB_E_CLASSNOTREG standard COM return value.

DIERR_EFFECTPLAYING
: The parameters were updated in memory but were not downloaded to the device because the device does not support updating an effect while it is still playing.

DIERR_GENERIC

An undetermined error occurred inside the DirectInput subsystem. This value is equal to the E_FAIL standard COM return value.

DIERR_HANDLEEXISTS
The device already has an event notification associated with it. This value is equal to the E_ACCESSDENIED standard COM return value.

DIERR_HASEFFECTS
The device cannot be reinitialized because effects are attached to it.

DIERR_INCOMPLETEEFFECT
The effect could not be downloaded because essential information is missing. For example, no axes have been associated with the effect, or no type-specific information has been supplied.

DIERR_INPUTLOST
Access to the input device has been lost. It must be reacquired.

DIERR_INVALIDHANDLE
An invalid window handle was passed to the method.

DIERR_INVALIDPARAM
An invalid parameter was passed to the returning function, or the object was not in a state that permitted the function to be called. This value is equal to the E_INVALIDARG standard COM return value.

DIERR_MAPFILEFAIL
An error has occurred reading either the vendor-supplied action-mapping file for the device or reading or writing the user configuration file for the device.

DIERR_MOREDATA
Not all the requested information fitted into the buffer.

DIERR_NOAGGREGATION
This object does not support aggregation.

DIERR_NOINTERFACE
The specified interface is not supported by the object. This value is equal to the E_NOINTERFACE standard COM return value.

DIERR_NOTACQUIRED
The operation cannot be performed unless the device is acquired.

DIERR_NOTBUFFERED
The device is not buffered. Set the DIPROP_BUFFERSIZE property to enable buffering.

DIERR_NOTDOWNLOADED
The effect is not downloaded.

DIERR_NOTEXCLUSIVEACQUIRED
The operation cannot be performed unless the device is acquired in DISCL_EXCLUSIVE mode.

DIERR_NOTFOUND
The requested object does not exist.

DIERR_NOTINITIALIZED
The object has not been initialized.

DIERR_OBJECTNOTFOUND
The requested object does not exist.

DIERR_OLDDIRECTINPUTVERSION
The application requires a newer version of DirectInput.

DIERR_OTHERAPPHASPRIO
Another application has a higher priority level, preventing this call from succeeding. This value is equal to the E_ACCESSDENIED standard COM return value. This error can be returned when an application has only foreground access to a device but is attempting to acquire the device while in the background.

DIERR_OUTOFMEMORY
The DirectInput subsystem couldn't allocate sufficient memory to complete the call. This value is equal to the E_OUTOFMEMORY standard COM return value.

DIERR_READONLY
The specified property cannot be changed. This value is equal to the E_ACCESSDENIED standard COM return value.

DIERR_REPORTFULL
More information was requested to be sent than can be sent to the device.

DIERR_UNPLUGGED
The operation could not be completed because the device is not plugged in.

DIERR_UNSUPPORTED
The function called is not supported at this time. This value is equal to the E_NOTIMPL standard COM return value.

E_PENDING
Data is not yet available.